Analysing the Effectiveness of Different Parameter Optimization Techniques for Complex Genetic Algorithms

by
Thomas Kielstra

A Thesis
presented to
The University of Guelph

In partial fulfilment of requirements
for the degree of
Master of Science
in
Mathematics

Guelph, Ontario, Canada

ABSTRACT

ANALYSING THE EFFECTIVENESS OF DIFFERENT PARAMETER OPTIMIZATION TECHNIQUES FOR COMPLEX GENETIC ALGORITHMS

Thomas Kielstra
University of Guelph, 2018

Advisor:
Dr. Daniel Ashlock

This thesis attempts to determine whether the genetic algorithm developed by Ashlock *et al.* (2014) could be modified to create hard to colour graphs for graph colouring algorithms, specifically the RS method. To determine whether this is possible, an optimal parameter setting for the genetic algorithm is obtained. This thesis analysed the edit commands, of the chromosomal regulatory sequence, to determine their general characteristics. Modified versions of particle swarm optimization and differential evolution were used to try to optimize the chromosomal regulatory sequence. This thesis shows that the edit commands do have some general characteristics, which could be used to modify the genetic algorithm. Both particle swarm optimization and differential evolution may be effective at optimizing the genetic algorithm given appropriate conditions. The genetic algorithm may be effective at finding hard to colour graphs for the RS method. This genetic algorithm could create hard graphs for other graph colouring algorithms.

Dedication

I would like to dedicate this thesis to my loving Grandmother, Tjaltje Dykstra, who joined our loving savour and LORD on December 1st, 2018.

Acknowledgements

I'd like to thank Dr. Ashlock for all his help and encouragement throughout this process. I would like to thank Dr. Demers, and Dr. Willms for their help as members of my advisory committee.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

The origin of Graph Theory is said to stem from the Seven Bridges of Königsberg problem. In Königsberg there is a river that travels around two islands, with seven bridges connecting the land masses seen in Figure 1.1. The problem asked if there was a way to create a walk in which you would cross every bridge once and only once. Leonhard Euler laid the foundations of graph theory when he proved it was not possible to create such a walk in 1736 (Euler, 1736). To prove this Euler created the first graph, seen in Figure 1.2, representing the land masses with vertices and the bridges with edges.

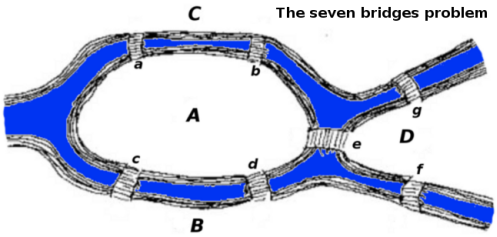


Figure 1.1: Königsberg Bridge (Ashlock, 2016)

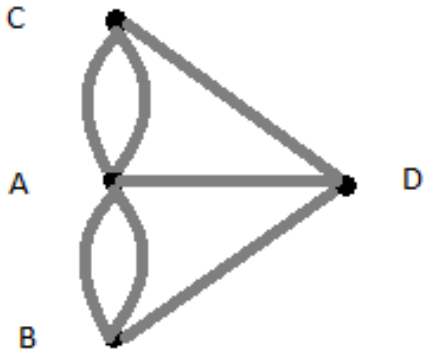


Figure 1.2: Königsberg Bridge Problem Graph

The origin of graph colouring is said to stem from a letter that Augustus de Morgan wrote to William Hamilton in 1852 (Kubale, 2004). The letter explained that one of his students, Francis Guthrie, had shown that the map of England only needed four colours to colour the

entire map, given that every county does not share a border with a county coloured the same colour (Kubale, 2004). The letter went on to pose the question: What is the minimal number of colours needed to colour a map (real or imaginary) on the plane such that no two bordering counties, land masses, are not coloured the same colour? (Kubale, 2004). In 1879, Arthur Cayley raised the problem at London Royal Society.

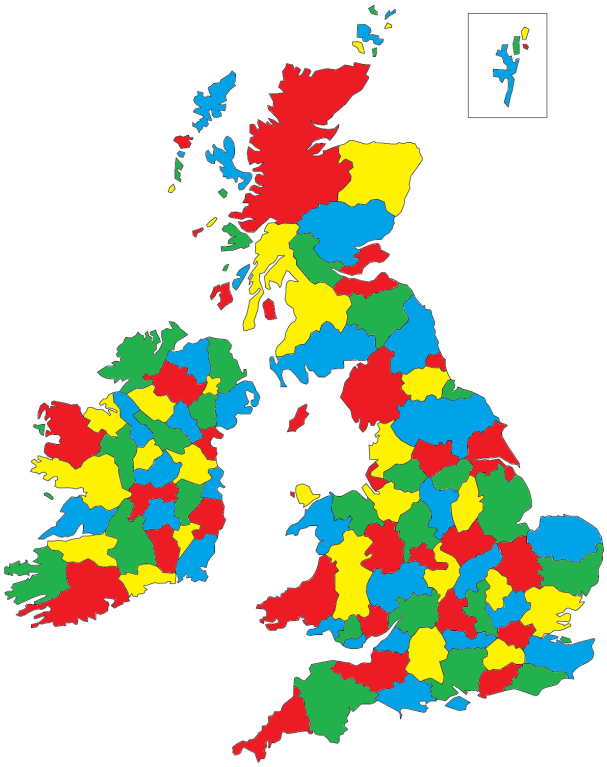


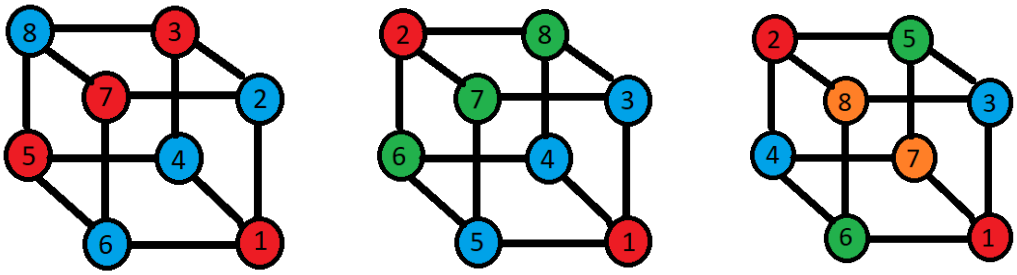
Figure 1.3: Map of UK Coloured with Four Colours (Jones, 2010)

The four colouring problem has been studied extensively since Cayley posed it in 1879. Alfred Kempe wrongly claimed to have solved it in 1879 (Kubale, 2004). In 1890, Heawood pointed out that Kempe's argument did not prove that at most four colours were needed to colour a map, but rather that at most five colours were needed to colour a map (Lint and Wilson, 2001). The four colouring problem was not solved until 1976 by Kenneth Appel and Wolfgang Haken (Lint and Wilson, 2001). This proof was widely considered experimental, since it required a large amount of computational aid. In 1997, Robertson *et al.* created a proof that required much less computation aid. Proving it without computer aid is still an open problem to this day, with many suggesting that it may never be solved; while others have suggestedd the non-computer aided proof is soon to come (Kubale, 2004). The four colouring problem is the most famous upper bound result in graph colouring (Kosowski and Manuszewski, 2004).

Similar to how Euler converted the Königsberg Bridge problem to a graph, maps coloured in this way can be represented by graphs in the following manner. The land masses, (counties, provinces, countries etc.) are represented by vertices, while an edge connects two vertices if there exists a border between them. The colouring of the map is represented by the colouring of the vertices. Incorporating the colouring restriction, detailed earlier, no two vertices that

are connected by and edge can be coloured the same colour. Such a colouring is called a *proper colouring*.

When properly colouring a graph, the order in which the vertices are coloured may affect the colouring of the graph. Suppose we colour the vertices of a graph in some order using an ordered set of colours and placing the smallest available colour, given the ordering, on each vertex as we encounter it. Using the cube as an example we can see that the cube can be coloured with 2 colours, 3 colours and 4 colours, depending on the order in which the vertices are coloured. The smallest number of colours needed to colour a graph is called the *chromatic number* of the graph.



(a) 2 Colouring (b) 3 Colouring (c) 4 Colouring

Figure 1.4: Three Different Colourings of a Cube

1.1 Graphs and Graph Colouring

1.1.1 Terms and Definitions

Graphs

DEFINITION 1 A *graph* G is a sets and multiset (V, E) where V is a finite set of vertices (also called nodes) and E is a multiset (elements are not necessarily unique) of 2-element unordered subsets of V, otherwise known as the multiset of edges or the multiset of nodal connections.

DEFINITION 2 The *order* (n) and *size* (m) of a graph are the number of vertices $(n = |V|)$ and edges $(m = |E|)$ respectively.

DEFINITION 3 Two vertices $u, v \in V$ are said to be *adjacent* if $(u, v) \in E$. One can think of u and v as directly connected.

DEFINITION 4 Traversing from vertex to vertex along edges is called *walking*. A walk is an ordered sequence of adjacent vertices $v_1, v_2, ..., v_k$. If the vertices in a walk are all unique then this walk is a *path*.

DEFINITION 5 The *length* of a path is length of the path's sequence or the number of edges in the path.

DEFINITION 6 *Cycles* are paths that connect a vertex to itself. The size of the cycle is the length of said path.

DEFINITION 7 A graph G is *connected*, when every pair of vertices are connected by a path. 1-vertex graphs are connected.

DEFINITION 8 The *distance between vertices* in a graph G, is the length of the smallest path connecting 2 vertices on a graph. This distance is denoted by $d(u, v)$, where $v, u \in V$. Note that $d(u, u) = 0$.

DEFINITION 9 The *eccentricity* of a vertex v is the largest distance any other vertex in the graph is away from it. Given $v \in V, eccentricity(v) = max\{d(u, v) \| u \in V\}$.

DEFINITION 10 A *directed edge* is a set of points that are connected by a line that can only be traversed in one direction, usually defined as an ordered pair (tail, head).

DEFINITION 11 A *loop* is an edge that connects a vertex to itself ($v \in V, (v, v) \in E$).

DEFINITION 12 A graph G is said to have *multiple edges* when there exists two vertices that are directly connected by more than one edge. (For $u, v \in V, (u, v)$ is not unique in the multiset E.

DEFINITION 13 A *simple graph* is a graph G that has no directed edges, loops, or multiple edges. A simple graph can be defined as a pair of sets (V, E) where V is a finite set of vertices, also called nodes, and E is a set of 2-element unordered subsets of V.

DEFINITION 14 A *complete graph* is a graph G where every vertex is connected to every other vertex in the graph.

DEFINITION 15 The *degree* of a vertex, $d(v)$ is the number of elements in V that are adjacent to that particular vertex v. $d(v) = |\{e \in E : v \in e\}|$. The degree of the vertex with the largest degree in a graph is $\Delta(G)$, while the degree of the vertex with the smallest degree in a graph is $\delta(G)$

DEFINITION 16 The *density* of a graph G is $g(G) = m/(\binom{n}{2})$

DEFINITION 17 Vertices v_1 and v_k are *path connected* if there exists a path from one vertex to the other.

DEFINITION 18 A *clique* V' in a graph G is a subset of V such that $\forall u, v \in V', (u, v) \in E$. A *maximal clique* is a clique that is not a proper subset of another clique. The *clique*

number $\omega(G)$ of G is the largest maximal clique in G.

DEFINITION 19 An *independent set* in a graph G is a subset $V' \subseteq V$, such that if $u, v \in V'$ then $(u, v) \notin E$. A independent set is *maximal* if it is not a proper subset of another independent set. The *independence number* of a graph is the size of the largest maximal independent set.

DEFINITION 20 A graph G is called a *k-partite* if the set of all its vertices can be partitioned into k subsets $V_1, V_2, .., V_k$, in such a way that any edge of the graph G connects vertices from different subsets. If a graph can be split up into 2 or 3 partitions that satisfy these requirements, then it is called at *bipartite graph* or *tripartite graph* respectively. If every point in each partition is connected to every point in the other partitions, it is called a *complete k-partite*. A complete k-partite is denoted by $K_{n_1,n_2,...,n_k}$, where n_i is the order or each partition. If the partition order is 1 for every partition is 1, then the k-partite is denoted as K_k.

DEFINITION 21 A *cut* partitions a graph into two disjoint sub-graphs. A *cut-set* is the set of edges that have one end point in each partition.

Graph Colouring

It is important to note that in this thesis graphs that are coloured are simple graphs.

DEFINITION 22 A *colouring function* c maps the vertices to the set of colours, which are the natural numbers ($c : V \mapsto \mathbb{N}$). Colouring functions partition graphs into independent subsets $V_1, V_2, ..., V_k$ for which, $V_i \cap V_j = \emptyset$ and $V_1 \cup V_2 \cup, ..., \cup, V_k = V$. Every element in V_i are coloured the same colour.

DEFINITION 23 A *proper vertex-colouring* (or proper colouring) of a graph G is a colouring such that any two points $u, v \in V$ are assigned different colours if $(u, v) \in E$.

DEFINITION 24 A graph G for which there exists a vertex-colouring that requires k colours is called *k-colourable*, while the colouring is called a *k-colouring*.

DEFINITION 25 The *chromatic number* ($\chi(G)$) of a graph G is the smallest number of colours needed to properly colour a graph. If $k = \chi(G)$, then G is called *k-chromatic*, while any colouring of G which requires $k = \chi(G)$ colours is called *chromatic* or *optimal*. Any colouring of this graph that uses more the k colours is a *suboptimal colouring*.

DEFINITION 26 The *saturation degree* $\varrho(G)$ of a vertex v in a coloured graph G is the number of colours used to colour the vertices adjacent to v.

DEFINITION 27 *Graph families* are sets of graphs with common structure or interesting

features. Certain algorithms look for graphs or subsets of graphs that are a part of particular families.

DEFINITION 28 An *r-regular* graph is a graph in which all vertices have a degree equal to r. A *cubic graph* is an equivalent term for a 3-regular graph.

DEFINITION 29 A *difference graph* G is a graph that has the vertices laid out is a ring, labelled 1 to n. The edges form a difference set, where the elements explain the connections between vertices.

The difference set used in this thesis is $\{\pm 1, \pm 2\}$ This means that the vertices are connected to the vertices two to the left and two to the right.

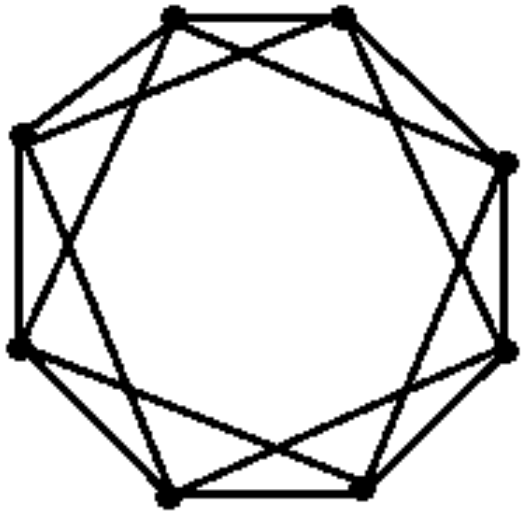


Figure 1.5: A Difference Graph with a set of $\{\pm 1, \pm 2\}$

1.1.2 Finding the Upper Bound of the Chromatic Number of a Graph

A large focus of graph colouring theory is finding the chromatic number for different graphs and classes of graphs. Trying to estimate the classical graph chromatic number is NP-hard when using either a k-relative or a k-absolute approximation algorithm (Kosowski and Manuszewski, 2004). Graph colouring can still be NP-hard, even with strong restrictions on the type of graph. Over time, many problems, like the four colouring of a tripartite graph (Kéry, 1964,; and the three colouring of a planar graph with degree 4 or less (Garey and Johnson, 1976), have been shown to be NP-hard. On the other hand, it has also been shown that for specific classes of graphs for example, interval graphs, the chromatic number can be determined using a simple formula, or its optimal colour can be determined in polynomial time (Kosowski and Manuszewski, 2004)(Nikolopoulos and Papadopoulos, 2000). Some have tried to create bounds on the chromatic number of a graph instead of trying to determine the chromatic number directly.

Finding effective bounds on the chromatic number has been hard, while the simplest bounds on graphs have proven difficult to use. It is trivial to see that, when colouring a graph, the smallest number the chromatic number ($\chi(G)$) could possibly be is one, while the largest number the chromatic number could be is the order of the graph ($1 \leq \chi(G) \leq n$). More bounds on the chromatic number have been discovered. The chromatic number has to be as large as the largest clique in the graph (($\omega(G)$), since a clique is a set of vertices in a graph where every vertex is connected to all the other vertices, in the clique. ($\omega(G) \leq \chi(G)$). This lower bound is not very useful, since finding the largest clique in a graph is also a NP-hard problem (Kozowski and Manuszewski, 2004). Moreover, Mycielski (1955) showed that a graph with a largest clique $\omega(G) = 3$ can have an arbitrarily high chromatic number. This means that one would have a lower bound, but no upper bound. Geller (1976) has proven a weaker lower bound on the chromatic number, which is much easier to calculate $\frac{n^2}{n^2-2m} \leq \chi(G)$, where n is the order of the graph and m is the size of the graph . Two known upper bounds are also used. Brooks (1941) showed that the chromatic number is less than or equal to the degree of the vertex with largest degree in the graph plus one, $\chi(G) \leq \Delta(G) + 1$. They showed that in order for $\chi(G) = \Delta(G) + 1$ the graph must have an odd cycle or the graph must be complete, otherwise the chromatic number is less than or equal to the degree of the vertex with the largest degree ($\chi(G) \leq \Delta(G)$) (Brooks, 1941). It has also been shown that $\chi(G) \leq \sqrt{2m} + 1$ (Kozowski and Manuszewski, 2004). In practice, the best upper bounds are obtained by using graph colouring algorithms (Kozowski and Manuszewski, 2004).

1.2 Algorithmic Approach to Graph Colouring

Graph colouring is a topic discussed by a wide variety of published theoretical papers. Since the 1970s it has been studied as an algorithmic problem, with many posing different types of graph colouring algorithms to solve specific types of graphs (Kabule, 2004). As discussed earlier, graph colouring problems are ofter NP-hard. Because of this, algorithmic approaches to colouring a graph that find minimal colourings that use more colours to colour a graph than the chromatic number of the graph, known as suboptimal graph colouring algorithm, have been developed and are being used widely for practical applications, like the travelling salesman problem (Garey and Johnson, 1976). Most of the classical graph colouring algorithms are sequential colouring algorithms that apply a greedy colouring. A greedy colouring is used because the minimal colouring of any graph can be obtained by

applying a greedy colouring.

DEFINITION 30 *Sequential colouring algorithms* colour graphs by first ordering the set of vertices into a finite sequence. This sequence is then coloured in this order using a greedy colouring. Sequential colouring algorithms are usually used in an attempt to find the chromatic number of a graph. The colours that are used are represented by $1, 2, 3, 4, ...$

DEFINITION 31 Given a sequence of vertices to be coloured, *Greedy colouring* is when the next vertex in the finite sequence is coloured distinctly from the vertices it is connected to, using the lowest available number. The largest number of colours that can be used to colour a graph using greedy colouring is called the *Grundy Number* or *Grundy Chromatic Number*.

Theorem 1 *The minimal proper colouring of any graph can be achieved by a greedy colouring.*

Proof: Assume without loss of generality, see below, that in the minimal colouring, each vertex with a given colour is adjacent to all of the smaller colours. Taking this colouring of the graph one could order the vertices such that all of the vertices coloured 1s come first, followed by the vertices coloured 2s second and continuing on in this fashion. When this sequence is colouring using a greedy colouring, all of the vertices that are coloured 1 in the minimal colouring will be coloured 1 via the greedy colouring. Since two adjacent vertices cannot be coloured the same colour, every vertex that should be coloured 1, will be coloured 1. Given that the first n colours have been coloured, the vertices that are going to be coloured the $(n+1)$st colour are adjacent to all of the smaller colours which have been coloured. This forces the greedy colouring to colour the vertex the $(n+1)$st colour. Thus, the resulting graph will be the minimal colouring.

Even though not all minimal colourings satisfy the requirement that every vertex coloured colour $k+1$ is adjacent to the first k colours, Campbell (2005) has shown that the graph can be recoloured in such a way that does not change number of colours used to colour the graph, but satisfies this requirement. It is important to note that even though it is possible to find the minimal colouring of a graph using a greedy colouring, it can be hard to do so, and even when you do, it is hard to prove that you have done so. Because of this uncertainty, when using colouring algorithms, there is a level of inaccuracy to every algorithm that is used.

Over the years of study a large number of problems have been solved using classical algorithms, in addition graph families have been developed to be used in practical applications

(Kozowski and Manuszewski, 2004). Over time, progressive suboptimal graph colour algorithms are developed to better colour certain types of graphs, each targeting specific types of graphs to overcome particular challenges faced by classical and current graph colouring algorithms. When using any graph colouring algorithm, it is important to know how accurate it is. There should be some sort of error bound on the results. Every algorithm was created with a specific goal in mind. This can usually be seen by how the creator decided to analyse the performance of their particular algorithm.

1.2.1 Analysis of a Colouring Algorithm's Performance

Comparing different colouring algorithms can be done in a variety of ways. How ever this is done, it is important to create simple criteria to compare the performance of colouring algorithms. Many different variables come into play when considering different graph colouring algorithms. The computational load (time required to compute) of the algorithms is also important to consider. If a particular algorithm outperforms previous algorithms, but takes a significantly longer time to solve any given problem, it might not actually be a better algorithm. For any suboptimal algorithm, a performance analysis is important, this could be a performance guarantee, but it could also be a fitness value comparison.

When using any colouring algorithm it is important to know how inaccurate the colouring algorithm can be. For large graphs, classical graph colouring algorithms tend to have very large computational load since graph colouring has an NP-hardness nature. In such cases, suboptimal colouring algorithms are used. When looking at a new suboptimal colouring algorithm, it is important to analyse the algorithm qualitatively and/or quantitatively to come up with a performance guarantee and/or fitness for certain sizes of graphs, as well as compare it to other colouring algorithms. For larger graphs, it may be more important to keep run time down, while for smaller graphs, accuracy may be more important. Depending on the particular size and shape of the graph, it is important to find the right balance between accuracy and computational intensity when using suboptimal colouring algorithms. Benchmarks or weak benchmarks are typically used when comparing different colouring algorithms.

DEFINITION 32 $C(G)$ is the number of colours used by the graph colouring algorithm C to colour a graph G.

DEFINITION 33 A colouring algorithm's *performance guarantee* $P(n)$ is defined by

$$P(n) = max\{C(G)/\chi(G) : G \text{ is a graph of order n}\}.$$

DEFINITION 34 A graph colouring algorithm's *fitness* $F(n)$ for a particular graph can vary. In this thesis the fitness function is defined by the average number of colours used to colour the graph by the graph colouring algorithms to colour a given graph divided by the minimum number of colours used to colour the same graph by the graph colouring algorithm .

$$\text{Given a graph } G,\ F(n) = ave\{C(G)\}/min\{C(G)\}.$$

DEFINITION 35 A colouring algorithm C is a *k-relative approximation algorithm* or simply a *k-approximation algorithm*, for a particular graph, if $C(G) \leq k \cdot \chi(G)$. The colouring algorithm C is called a *k-absolute approximation algorithm*, for a particular graph, if $|C(G) - \chi(G)| \leq k$.

Every colouring algorithm has certain strengths and certain weaknesses when it is applied to certain graphs. Sometimes, when a colouring algorithm is applied to a graph it optimally colours the graph, but it can also colour a graph suboptimally.

DEFINITION 36 When applying a colouring algorithm to a graph G. The graph is *slightly-hard-to-colour* (**SHC**) if there exists at least one way of applying the algorithm that results in a suboptimal colouring. The graph is a *hard-to-colour* (**HC**) if the colouring is suboptimal no matter how you apply the colouring algorithm.

Even though there are clear lines for when a graph is slightly-hard-to-colour and when a graph is hard-to-colour, there is a continuous spectrum for the *hardness* of a graph. In this thesis the hardness of the graph will be determined by the fitness of the graph colouring algorithm's fitness when applied said graph.

DEFINITION 37 A *family of graph colouring algorithms* $\mathbf{C}$ is a set of algorithms that are similar or are being compared. $\mathbf{C} = \{C_1, C_2, ..., C_k\}$.

DEFINITION 38 A graph is a *benchmark* for a family of graph colouring algorithms $\mathbf{C}$ if it is HC $\forall C_i \in \mathbf{C}$. It is a *weak benchmark* if it SHC $\forall C_i \in \mathbf{C}$.

When determining which graph colouring algorithm is best for a specific use, each graph colouring algorithm under consideration must be evaluated using a fitness function. Generally, when comparing different colouring algorithms, every algorithm must be applied multiple times to the graph to figure out the algorithm's true effectiveness. Graph colouring algorithms have been compared using the following criteria: their performance guarantee; the size of the smallest hard to colour graph; their ability to colour graphs that are hard to colour for both algorithms.

In order to improve a graph colouring algorithms, one must find graphs that are hard to colour for a graph colouring algorithm. Finding simple hard to colour graphs has turned out to be a very difficult problem. Ashlock and Barlow (2015) have used evolutionary algorithms to create small hard-to-colour graphs to use when comparing different graph colouring algorithms. Using evolutionary algorithms to find these hard to colour graphs will be one of the focuses of this thesis.

1.3 Evolutionary Computation

Using evolutionary based automated techniques to try and solve problems was originally proposed by Alan Turing in 1950. In the 1960s Nils Aall Barricelli started simulating evolution using evolutionary algorithms and artificial life, which Fraser (1958) built upon by simulating artificial selection. Ingo Rechenberg (1973) showed that evolutionary strategies could be used to solve more complicated engineering problems. Holland (1975) introduced a new type of evolutionary algorithm called a genetic algorithm. As computers continue to develop, evolutionary computation has become more applicable to a wider variety of applications.

In evolutionary computation, convergence does not have same rigourous usage as it does in traditional mathematics.

DEFINITION 39 *Evolutionary computation convergence* (EC-convergence) is a loss in diversity in the population, or approaching a maximal solution for the algorithm.

Just like the lack of rigour in the definition of EC-convergence, the idea of EC-convergence is not used rigourously in evolutionary computation. In this thesis, EC-convergence will be used in a non-vigourous manor.

1.3.1 Evolutionary Algorithms

Evolutionary algorithms are considered to be a subset of evolutionary computation. Evolutionary algorithms are metaheuristic optimization algorithms that analyse given populations using mechanisms that simulate the biological evolutionary processes of mutation, selection, reproduction and recombination. Soïesen (2013) defines two classes of algorithmic solvers: exact algorithms and heuristics. An exact solver guarantees a solution in a given time, while a heuristic has no such guarantee (Soïensen, 2013). Exact solvers find the exact

solution using exhaustive method, which in some cases have too have to much of a computational load to be practical. In such cases, heuristic solvers are beneficial, because they tend to have less computational load. Soŕesen defines a metaheuristic as a program that is designed to develop its own heuristic and then apply it to find the best solution that it can.

The evolution in the evolutionary algorithms occur as heuristic mechanisms are implemented repeatedly. When using evolutionary algorithms, it is essential to determine useful evolutionary mechanisms and fitness functions. Evolutionary algorithms evolve in a predetermined way designed to improve its population's fitness as it evolves.

DEFINITION 40 The *population* is a set of possible solutions or a singular solution to the problem.

The fitness of the population is dependent upon the fitness of the individuals of the population, which is determined using a fitness function. Some evolutionary algorithms will abort any evolution that does not increase the fitness of the population, some will abort any evolution that decreases the fitness of the population, while others will let the population continuously evolve for a specific number of iterations relying on the design of the evolutionary algorithm to generate a fit population.

Ashlock (2018) explains that evolutionary algorithms have a biologically based structure, made of nine parts (1-9). The first five parts of an evolutionary algorithm define the problem and how it is analysed. Parts six through eight combine to form the iterative portion of an evolutionary algorithm, defining how the algorithm will evolve the population. The final part of the algorithm explains how the algorithm stops evolving the population. These algorithms are typically used to effectively search a solution space for an optimal solution.

The first five parts help define what the problem is that is being analysed and how it is analysed. Every evolutionary algorithm has a problem (1) that is designed to solve and a way to *represent* possible solutions on a computer (2). The problems being tested by evolutionary algorithms are designed to represent a real world problem by using a model of some sort, similar to how Euler created a graph to represent the Könnigsberg bridge problem. The problems being tested may also be the real problem that is being tested; for example, if you were testing graphs you would use a graph. Epidemics have been represented by graphs (Ashock and Shiller, 2011). Every algorithm has a way in which it *generates an initial poputlation* (3) and a predetermined *size of the evolving populations*(4). In order

for the evolutionary algorithms to evaluate members of its population, there is an *objective function* (5), often called a fitness function, that is capable of measuring the quality of a solution.

DEFINITION 41 The *fitness function* is a rule that gives each member of the population a value.

The first five parts layout the *environment* of an evolutionary algorithm.

Once the environment has been defined, the iterative process of the evolutionary algorithm can be defined (parts (6)-(8)). Evolutionary algorithms rely on mechanisms that are derived from evolutionary processes like selective breading, mutation and genetic variation, and natural selection. In order to evolve a population, evolutionary algorithms use a *selection scheme* (6) to pick solutions that encourage better results, based on the objective function that is used. The selection scheme picks the member of he population to be parents, using them to create the new generation. This is similar to selective breeding.

DEFINITION 42 A *parent* is a member of the population, a possible solution, that is used to create the next generation of solutions.

A *variation operator* (7) is used to generate new solutions from an old solution or old solutions. The variation operator explains how the parents are used to generate the next generation of solutions. These operators could cross-breed parents, or a single parent could be mutated to create new solutions. The operators mimic mechanisms of biological evolution.

These new solutions are integrated into the population using a *replacement scheme* (8). Replacement schemes are also based off of the objection function, encouraging a fitter population. Different replacement schemes drastically change the effectiveness of the algorithms. This part of evolutionary algorithms is analogous to natural selection.

The final part of the evolutionary algorithm defines when the algorithm will terminate. Evolutionary algorithms, like most interactive algorithms, will not stop evolving until they reach their *stopping criterion* (9). This could be a number of iterations or a fitness for the most fit member of the population. The solution of the evolutionary algorithm is the usually the member of the final population that is the most fit. Most evolutionary algorithms that are ineffective at searching the solution space have not properly formulated one or more of the nine parts of the evolutionary algorithm.

Solution spaces are usually analysed like topographical maps, where elevation is deter-

ministic of the fitness of a particular solution. If the solution is more fit, based on the fitness function in the evolutionary algorithm, it will have a higher elevation. If the solution is less fit, it will have a lower elevation. How far two different possible solutions are from each other will be analogous to how different the characteristics of these solutions are. It is important to note that these maps are theoretical constructs and are discussed in the abstract sense.

1.3.2 Genetic Algorithms

Genetic algorithms are a subset of evolutionary algorithms. What makes a genetic algorithm a genetic algorithm is its population. Like the name suggests these evolutionary algorithms' population members are analogous to chromosomes that are found in living species, only in a simpler form. Each member of the population P can be represented by a sequence

$$P = \{p_1, p_2, p_3, ..., p_n\}$$

The members of the population are evolved using mutation mechanism that are based on genetic mutation, like cross point mutations and point mutation. Selective processes like crowding replacement or tournament selection are used as well. In this thesis the population members are generated and regulated by a *chromosomal regulatory sequence*.

DEFINITION 43 A *chromosomal regulatory sequence*, also know as a parameter probability density, determines the likelihood that each type of operator will appear in a chromosome.

The way a population member is used by the genetic algorithm is unique to the genetic algorithm. Following the theme of genetic evolution, the chromosomes evolve, using breeding and mutations.

Genetic algorithms were studied theoretically until the mid 1980s. It was not until The First International Conference on Genetic Algorithms, which was hosted in Pittsburgh Pennsylvania, that genetic algorithms started to be applied to different situations. As genetic algorithms have been used more frequently, the computational load of the genetic algorithms have increased, but they are all still built using the same basic structure.

1.4 Conclusion

Graph colouring has been a topic of great discussion because of its apparent simply underlining problem, but relatively hard computation. As stated earlier graph colouring

has been treated as an algorithmic problem since the 1970s, with an emphasis on trying to bound the chromatic number of different graphs. As different algorithmic techniques arise and more theory is developed on graphs, more graph colouring algorithms are developed, further pushing the applications of graphs and graph colouring.

Chapter 2

Literature Review

Graph theory and the applications of evolutionary algorithms have become wide areas of research with a lot of diverse applications. As computers have developed, this has only increased. As stated earlier, the first computer aided proof was the four colouring problem, solving one of the first graph colouring theory problems. Graph colouring theory has continued to develop, being treated as an algorithmic problem since the 1970s. Since the 1970s, graph colouring algorithms have become increasingly diverse using a wide variety of techniques. They are being applied in a wide variety of ways. Trying to find the chromatic number of a graph is still actively studied today.

Evolutionary algorithms have continued to develop. As they have become more advanced, different types of evolutionary algorithms have evolved. The different types of evolutionary algorithm are categorised by similar attributes or mechanisms shared by the algorithm. One type of evolutionary algorithm that has been developed is called a genetic evolutionary algorithm or a genetic algorithm. Genetic algorithms have been shown to be useful, but tricky to implement, since they so heavily rely on their chromosomal regulatory sequences. Different parameter optimization techniques have been developed, but as genetic algorithms become more computationally intense, these methods have become ineffective at optimizing the chromosomal regulatory sequences.

2.1 The Development of Graph Colouring Algorithms

Using graph colouring algorithms to try and find the chromatic number of a graph has been done since the 1970s. Most graph colouring algorithms that are developed stem from finding weaknesses in old graph colouring algorithms. These old graph colouring algorithms

are modified in particular ways to try and overcome these weaknesses. The graph colouring algorithm may be ineffective when trying to find finding proper colourings that are k-relatively or k-absolutely close to the chromatic number of the graph or it might take too long to fined these proper colourings. Classical colouring algorithms are generally older and more simplistic, using one or two heuristics; while modern colouring algorithms are typically have more computational load, in an attempt to colour more difficult graphs, using multiple, more complicated, heuristics. These modern graph colouring algorithms typically attempt to overcome the weaknesses of previous graph colouring algorithms.

New graph colouring algorithms are still being developed, usually to find effective graph colouring algorithms that are able to better colour particularly difficult graphs that have challenged other colouring algorithms, classical or modern. As graph colouring algorithms become more and more refined, finding graphs that are hard to colour for these graph colour algorithms has proven to be a difficult task. This thesis will look at whether genetic algorithms can be used to create graphs that are harder for these graph colouring algorithms to colour.

2.1.1 Classical Colouring Algorithms

A graph colouring algorithm is considered classical if it was developed in the 1960s, 1970s or early 1980s and if its heuristics are simplistic. Most of the classical colouring algorithms are called methods. Most classical graph colouring algorithms are sequential colouring algorithms that purpose a way to order or reorder the vertices in the finite sequence to decrease the number of colours used when applying a greedy colouring. When looking at modern graph colouring algorithms and evolutionary graph colouring algorithms, it is important to understand how the classical colouring algorithms work and some of their strengths and weaknesses, since most modern graph colouring algorithms modify these basic heuristics and the evolutionary graph colouring algorithms tend to use these basic heuristics.

The *naive* method, also known as the *random sort* (RS) *method*, uses one of the most basic methods to order the vertices. It sorts the vertices by randomly ordering them in a sequence and then greedily colouring the sequence (Kosowski and Manuszewski, 2004). It should be noted that the random ordering method has no hard to colour graph since it is possible to randomly assign a sequence that would colour the graph optimally with a greedy colouring algorithm. Even though it is possible to find a chromatic colouring using the RS method, these colouring may be very rare, meaning that the likelihood of finding these

colourings are incredible low. The *connected sequential* (CS) *method* orders the sequence by randomly choosing vertices that are connected to vertices already in the sequence. If a vertex is already in the sequence, it cannot be picked again. This approach can help prevent suboptimal colourings in many situations (Kosowski and Manuszewski, 2004).

One of the oldest non-random sequential ordering methods is the *largest first* (LF) *method*, introduced by Welsh and Powell in 1967. They noted that vertices that had smaller degrees have more flexibility than vertices with larger degrees. They argued that the vertices with larger degrees should be coloured first, while lower degree vertices should be coloured later. When using the LF method, construct a sequence by ordering the vertices of the graph by the vertices' degree, from largest degree to smallest degree. Then apply greedy colouring to the sequence. The LF method has a run time of $\mathcal{O}(m+n)$, where m and n are the size of the graph and the order of the graph, respectively (from DEFINITION 2. The LF method has a worst possible $n/2$-relative approximation for any given graph (that is $k \leq n/2$ in Definition 33). This means that given a graph of order n, the colouring with the largest number of colours will be $n/2$ time the chromatic number. Unless the Chromatic number is 1, this means that it is bounded by a least n, meaning it is bounded by the order of the graph. This is not an effective bound on the colouring of the graph. The performance guarantee is linear (Kosowski and Manuszewski, 2004).

In 1972, Matula *et al.* noted that the LF *method* could be improved if instead of colouring the largest first, one colours the smallest last. This is known as the *smallest last* (SL) *method*. He noted that by doing so you could create subgraphs that could be coloured optimally with little connectivity to the larger graph. The main difference between the LF method and SL method, is the generation of the sequence that determines the order in which the vertices are coloured. In the SL method the sequence is generated choosing a vertex with the lowest degree repeatedly until all of the vertices have been chosen. When a vertex is chosen it is removed from the graph, decreasing the order of all of the vertices that are adjacent to that vertex by one. Once all of the vertices have been selected, the sequence is then inverted. Just like in the LF method, once the SL method has created the sequence in which the vertices will be coloured, a greedy colouring is applied to said sequence. The SL method is able to optimally colour many graphs that the LF method had difficulty with, e.g. Johnson's graphs and Mycielski's graph as well as trees and cycles (Kosowski and Manuszewski, 2004). The SL method's performance is similar to the LF method. The SL method has a run time $\mathcal{O}(m+n)$. The SL method's worst possible *k-relative* approximation is slightly better than the LF method's, $k \leq n/6$ instead of $k \leq n/2$. This is an improvement, since it will limit

the worst colouring to less than the order of the graph if the chromatic number is less than 6. It is slightly better but still not very good. The SL method's performance guarantee is also linear (Kosowski and Manuszewski, 2004).

Matula *et al.* (1972) also introduced a way to improve the LF method, the SL method and the RS method, called the colour interchange method. This method does increase the runtime of all three algorithms. Whenever a new colour has to be used, the colour exchange algorithm will try to exchange previously coloured vertices to try and free up a colour that has already been used. When the colour interchange method is applied to them, the LF, SL and RS methods are known as the LFI, SLI, and RSI methods respectively (Kosowski and Manuszewski, 2004). The runtimes of all three methods, when the colour interchange method is applied is $\mathcal{O}(mn)$ instead of $\mathcal{O}(m+n)$. Their performance guarantees are still linear.

In 1974, Johnson suggested the *greedy independent sets* (GIS) *method*. This method starts with a vertex and colours it in the usual way. It then randomly selects a vertex that is not adjacent to any of the vertices already coloured this colour and colours it the same colour. It does this until it is no longer able to pick a new vertex and colour it the same colour. It then removes these vertices from the graph and repeats the process on the remaining subgraph. The number of iterations required to attain the empty set is the number of colours used to colour the graph. The GIS method runs in $\mathcal{O}(mn)$ time, with a performance guarantee $\mathcal{O}(n/log(n))$ (Kosowski and Manuszewski, 2004).

In 1979, Brélaz introduced the *saturated LF* (SLF) *method* under the name DSATUR (degree saturation). He made the observation that the saturation degree of the vertex being coloured constrains the graph colouring, not the degree of the coloured vertex. The first vertex is chosen by selecting a vertex from the graph of vertices that have the highest degree. Once, the first vertex has been coloured, the SLF method works by finding the vertex with the highest saturation degree and greedily colouring it. It continues to do this until it is completely coloured. The SLF method has a $\mathcal{O}((m+n)log(n))$ run time (Turner, 1988) and a linear performance guarantee (Kosowski and Manuszewski, 2004). This algorithm can optimally colour nearly all k-chromatic graphs (Turner, 1988).

As these algorithms have been tested over time, graph colouring has been used for more specific applications. These applications have larger and more complicated graphs that provide challenges for classical colouring algorithms. These challenges have lead to the development of new modern graph colouring algorithms.

2.1.2 Modern Colouring Algorithms

Modern graph colouring algorithms usually look at colouring specific types of graphs effectively. They combine multiple heuristics when colouring the graph. These heuristics are designed to capitalize on the graph's composition, like the eccentricity of a vertex in a graph or symmetries that lie within the graph. These modern graph colouring algorithms look beyond just the qualities of individual vertices of the graph that is being coloured.

There have been many improvements for specific types of graphs. Goldberg *et al.* (1988) showed that breaking the symmetry in parallel was effective for solving problems on sparse graphs, like planar graphs, which they were able to colour with 5 colours. Schneider and Wattenhofer (2010) showed that using a technique they call *using multiple trails* can help by breaking symmetries for distributed algorithms.

Modern graph colouring algorithms must balance effectiveness with computational intensity. Barenboim and Elkin (2009) have discussed the trade-off of time verses effectiveness of graph colouring algorithms, as well as breaking the Szegedy and Vishwanathan threshold. They admit that the Szegedy and Vishwanathan (1993) conjecture, which states that graph colouring algorithms may not achieve a run time less than $\mathcal{O}(\Delta(G)log(\Delta(G)))$, may still be true. $\Delta(G)$ is the degree of the vertex with the largest degree in the graph (DEFINITION 15).

Modern algorithms have been developed that do not try to find the chromatic number of a graph directly but find bounds on the chromatic number. Kuhn (2009) has created a colouring algorithm that can deterministically compute a $\Delta(G)+1$ colouring of a graph and compute the maximal independent set of a graph in a run time of $\mathcal{O}(\Delta(G)+log(n))$ besting Barenboim and Elkin's (2009) algorithm, which Kuhn and Wattenhofer (2006) showed had a run time of $\mathcal{O}(\Delta(G)log(\Delta(G))+log(n))$. Schneider and Wattenhofer (2008) have presented multiple algorithms that help solve the maximal independent set problem on growth-bounded graphs.

2.2 Evolutionary Algorithms

Evolutionary algorithms have been used on graphs for a variety of different applications. Cutello *et al.* (2003) and Galinier and Hoa (1999) have used evolutionary algorithms to try and find proper colourings of combinatorial graphs. Combinatorial graphs are common problems that arise naturally from scheduling problems. Alhomdi and Reed (2013) and Kumar

et al. (2007) have applied evolutionary algorithms to try and locate the cut-sets of a graph. Garcia-Piquer *et al.* (2014) and Pizzuti (2012) have used evolutionary algorithms to cluster a graph into tightly coupled sub-graphs. In 2006, Ashlock wrote *Evolutionary Computation for Optimization and Modeling,* which has led to a lot of research on how evolutionary algorithms can be used to help develop new graphs and graph colouring algorithms for different applications. Research has also been done on how effective evolutionary algorithms can be used to replicate the spread of epidemics (Ashlock *et al.* , 2011)(Ashlock and Timmins, 2016)(Timmins and Ashlock, 2017).

Since graphs have been used in a variety of applications, it is not surprising that different shapes of graphs are used for different applications. Ashlock *et al.* (2014) have developed a way to create graphs, with desired qualities for different applications, mainly focused on epidemic populations. Since these evolutionary algorithms have been shown to be able to find graphs with specific qualities, one question that could be asked is whether evolutionary algorithms could be used in graph colouring theory to find harder-to-colour graphs for a particular graph colouring algorithm; or more specifically, whether the genetic evolutionary algorithm, developed by Ashlock *et al.* (2014) could be used to find harder-to-colour graphs for graph colouring algorithms.

2.3 Genetic Algorithms

Genetic algorithms, created by John Holland in the 1960s and 1970s, mimic the process of mutation to create and modify the populations of the algorithms (Koza, 1998). In 1975, De Jong put a considerable effort into optimizing the control parameters of evolutionary algorithms by using exhaustive search methods. He found control parameters that were optimal for his genetic algorithm. In 1986, Grefenstette showed how genetic algorithms can be used to optimize complex systems. He showed optimization genetic algorithms could be improved, by using another genetic algorithm to fine tune the variation parameters of the optimization algorithms. In 1989, Goldberg and many others, thought genetic algorithms were robust problem solvers, showing similar performance when used to try and solve a variety of problems (Goldberg, 1989). By 1996 however, Michalewicz and Schoenauer (1996) acknowledged that in order for genetic evolutionary algorithms to be applied effectively to specific applications, specific setup is required. De Bonet *et al.* (1997) showed that by analysing the landscape of the solution space, one might be able to create a directed random search of the parameters, significantly increasing the speed in which optimal parameters are

found.

2.3.1 Parameter Optimization

Eiben *et al.* (1999) explained the difficulty with trying to optimize genetic evolutionary algorithms. The parameters, though operating distinctly, are not independent, and systematically trying all different combinations and parameter tuning is too time consuming to be practical, even if significant effort is made while setting the parameters. Eiben *et al.* (1999) explained that even though there have been a few theoretical investigations on the optimal population size and operator probabilities, the application of these results is very limited. Because evolutionary algorithms are intrinsically dynamic and adaptive, using rigid parameters that do not adapt contrasts the very nature of the evolutionary algorithm, as Eiben *et al.* (1999) explain. The different parameters function differently, and the way in which they search the solution space is also different. This disparity between operators means that one parameter may be very effective early in the evolution, while having negative effects in the late stages of evolution; other parameters may be ineffective early in the evolutionary process, but effective in the late stages of the process (Eiben *et al.* , 1997).

Much research has been done to try and improve the performance of genetic algorithms. Ballester and Carter (2003) looked at how different types of evolution mechanics and populations effect the performance of genetic algorithms. They showed that genetic algorithms with random selection of parents and crowding replacement were more robust than genetic algorithms that used tournament selection of parents or random replacement. Random selection is where the parents are selected randomly; while tournament selection selects the parents by randomly selecting multiple members of the population to be possible parents and then having the members with the highest fitness become the parents for the next generation (Miller and Goldberg, 1995). Crowding replacement is done by comparing edit chromosomes similarity members of the population and allowing the more fit member to survive; while random replacement randomly selects which members to discard (Galan and Mengshoel, (2010). Many different methods have been developed to try and optimize different parameters over the years, but this research is still young with a variety of different approaches being compared. Recently particle swarm optimization and differential evolution have become popular techniques, when analysing solution spaces of genetic algorithms. This thesis compares the performance of particle swarm optimization, differential evolution and a new technique that is similar to a beam search on a bounded simplex.

Particle Swarm Optimization

Kennedy and Eberhar (1995), Shi and Eberhar (1998) and Kennedy (1997) first designed particle swarm optimization in an attempt to simulate social behaviours of birds and fish. It has four parts: initializing the population; evaluating the population; evolving the population; and ending the particle swarm optimization. The members are called particles and they swarm around the solution space to try and find optimal solutions, hence the name. These particles have a location, performance, and the location of the best previous location the particle visited. The location of the global best location for all of the particles is also stored. These are used to help evolve the population. Shi and Eberhar's (1998) particle swarm optimization is used to optimize parameters over a defined region of an unbounded continuous domain. The population is usually initialized by taking a uniform random sample of the defined region to determine the location of each of the particle in the population. These particles are also given a random velocity. The number of particles used (population size) is specific to each particle swarm optimization. Particle swarm optimization uses the location of the particle to determine the performance of the particle.

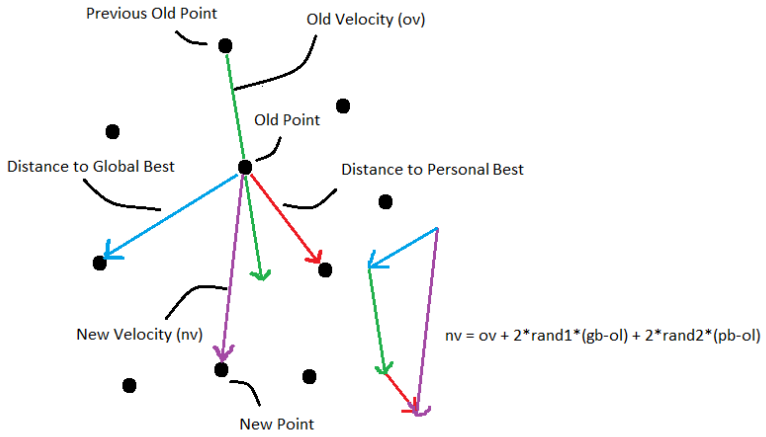


Figure 2.1: An Example of a Movement of a Particle in a Particle Swarm Optimization

The population evolves by updating the locations of the particles (Figure 2.1). Shi and Eberhar (1998) solved for the new location (nl) of the particle by taking the old location (ol) and adding a velocity vector (vv) based on the old velocity (ov) of the particle, the personal best location (pbl) of the particle, and the global best location (gbl) of all the particles used by the genetic algorithm.

$$nl = ol + vv, \quad \text{where} \quad vv = ov + 2 * c1 * (pbl - ol) + 2 * c2 * (gbl - ol)$$

The variables c1 and c2 are chosen by taking a uniform random sample of the interval [0,1], for each iteration of each of the particles. The particle swarm optimization usually ends after a certain number of evolutions.

Hanchate and Bichkar (2014) and Montalvo *et al.* (2010) have tried using particle swarm optimization with some success. Montalvo *et al.* (2010), tried using a variant of particle swarm optimization designed to require no a priori parameter tuning. Particle swarm optimizations have been widely used. The applications and theoretical uses of particle swarm optimizations have been analysed by Poli (2008) and Bonyadi and Michalewicz (2017). As genetic algorithms have become increasingly computationally intense, applying any form of parameter optimization to these genetic algorithms takes an increasingly long time.

Differential Evolution

In the 1990s, Kenneth Prince created an algorithm called differential evolution in an attempt to solve the Chebychev polynomial fitting problem (Storn, 2013). Storn and Prince (1997) first introduced differential evolution in 1997, showing how it can be used for global optimization in continuous systems. Differential evolution has four main parts: initializing and evaluating the initial population; creating and evaluating the mutant generations; selecting the new generation; and ending the differential evolution (Storn and Prince, 1997). The members of the population for differential evolution are called agents. Differential evolution was created, just like particle swarm optimization, to optimize parameters on a defined region of an unbounded continuous domain. The original population member's location is determined by taking a uniform random sample of the defined region being analysed. The agent's location's performance is determined by the objective function of the genetic algorithm. These agent's locations are stored in a chromosome. The number of agents used (size of population) is unique to the differential evolution.

The mutant population is created by using the locations of the original population. A mutant chromosome is created. One mutant agent is created for every agent in the original population (Figure 2.2). The location of the mutant agent(x_m for the agent x is determined using other member of the population.

$$x_m = a + F * (b - c)$$

The variables a, b and c are distinct locations of distinct agents from the current population. They are also not equal to the location of agent x. The variable F is a selective factor, which

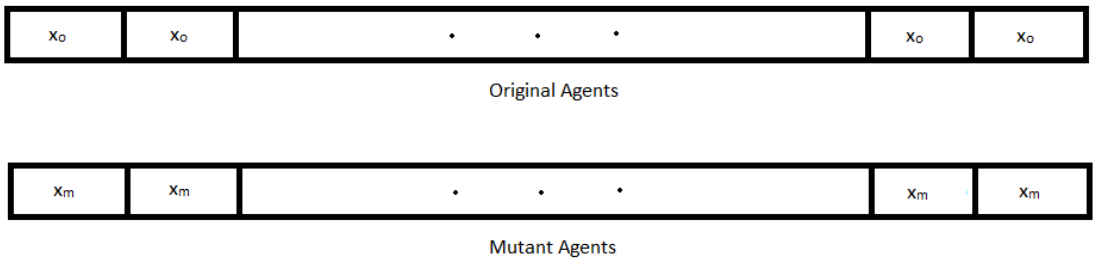


Figure 2.2: Mutant Chain and Original Chain

is randomly selected between [0.5,1] (Storn, 2013). Once the location of the mutant agents have been determined, their performance is determined, using the same process from the original agents.

The next generation of agents is selected by taking the best of the original agent or its mutant agent. This forms the new original chromosome for the next generation. Differential evolution, just like particle swarm optimization usually ends after a certain number of iterations (Storn, 2013). For this thesis, since chromosomes are used by the genetic algorithm, these chromosomes will be referred to as chains. Every link in the chain will represent an agent.

In 2008 Zhang *et al.* showed that differential evolution could be applied to discrete spaces, given there is a function applied that maps the discrete solutions to a continuous space. Lichtblau (2012) showed how Differential Evolution can be used in Discrete Optimization, specifically talking about its use in parameter optimization of genetic algorithms. Storn and Prince (1997) show that Differential Evolution is a simple and fast evolution, which would make it a less computationally intense method of optimizing the parameters of a genetic algorithm.

2.3.2 Beam Search

Effectively searching memory, or finding possible solutions to a problem has been a goal of computers since Alan Turing tried to crack the Enigma machine in the Second World War. In 1977, Reddy coined the term beam search to describe a greedy tree search algorithm that searches a large graph quickly. Each node of the graph can be evaluated to determine its performance. Beam search is often used when translating a phrase from one language to another (Freitage and Al-Onaizan, 2017). In translation, beam search operates by finding the most

probable word or words in the new language for the first word in the sentence from the old language. It then removes all the other words as possibilities for the first word in the sentence. Beam search then finds the most probable word or words in the new language for the second word in the sentence in the old language, given that the first word that is chosen. Beam search continues in this way until it finishes the sentence it is trying to translate. Freitage and Al-Onaizan (2017) analyse beam searches that keep more that just one branch at each evaluation, and remove the branches that perform worse later. These analyse how to determine what is the effective number of branches to keep from word to word. When only one solution is kept and used to make the next evaluation, it is called a greedy beam search.

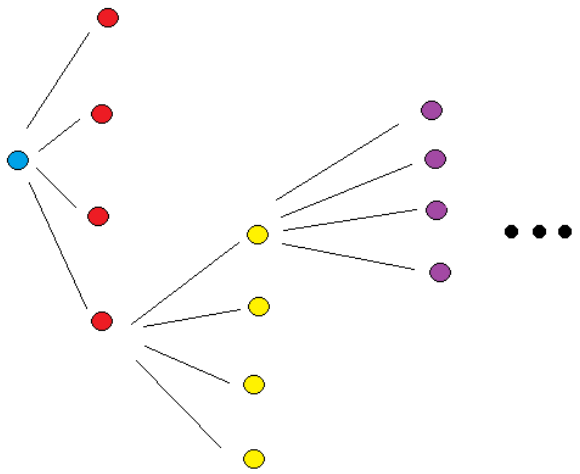


Figure 2.3: Greedy Beam Search

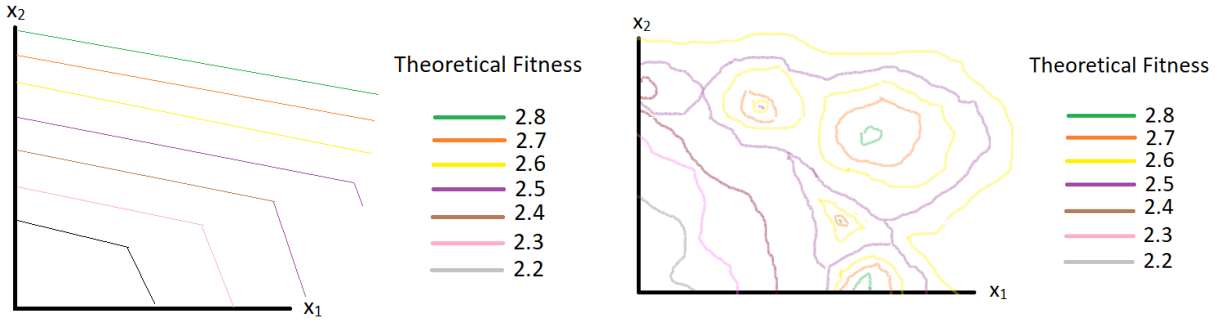
The graph that beam search is searching for translation is every possible sentence in the language it is translating to. Beam search evaluates the different branches and chooses the best one. It then continues to search by searching only using the best branch. This thesis looks at whether beam search could be used to optimize the chromosomal regulatory sequence of the genetic algorithm. It will be called beam optimization.

LASSO Regression

Recently in statistics, different regression techniques have been developed not to explain the impact that different variables have, but to find or approximate the solution to an optimization problem given a limited amount of input. These methods have resulted from modifications made to the least square regression technique, first clearly explained by Legendre in 1805.

Least Squares Regression creates the best linear unbiased estimation of the solution space, creating a linear function of the different variables. Least squares regression works best when each variable is independent and unbounded, meaning that there are no constraints on any of the variables. When there is some covariance between the variables, least square regression does not work effectively. If the solution space was a topological map with a surface

$z = f(x_1, x_2)$, the level curve would be simple and easy to understand like Figure 2.4(a). When the variables are not independent from each other, the surface of the topological map z becomes very complex. Similar results would be found in different locations in the solution space, seen in Figure 2.4(b). There is no longer a simple function that is able to explain the relationship between the variable. The large variability in the objective function of the genetic algorithms causes too many local extrema for least squares regression to be effectively used.

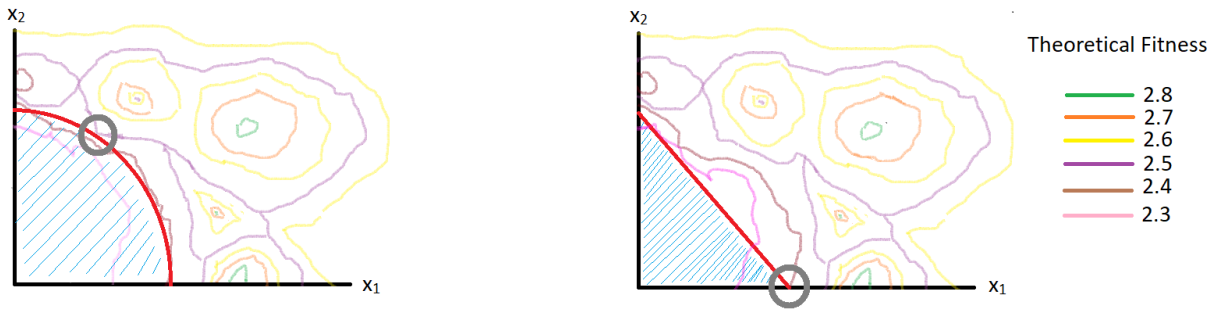


(a) Independent Variables

(b) Not Independent Variable

Figure 2.4: Possible Example of Level Curve

Ridge regression was one of the first regression techniques developed to try and deal with variables that were not independent or linear, invented independently in a wide variety of contexts. Tikhonov (1943) and Phillips (1962) used it to try and solve integral equations. Hoerl expounded the finite dimensional case in 1962, and since his paper in 1970, the term ridge regression has been used. The simplest explanation of Ridge regression is that it estimates the largest possible solution, given the variables used are within a certain distance from the origin, using a 2-norm (Figure 2.5(a)).



(a) Ridge Regression

(b) LASSO Regression

Figure 2.5: Ridge vs LASSO Regression

In 1996, Tibshirani introduced LASSO (least absolute shrinkage and selection operator) regression by making an adjustment to the distance measurement, using a Manhattan distance instead of a Euclidean distance (Figure 2.5(b)). This adjustment meant that there was no longer an analytical solution. Through numerical methods a nearly optimal solution can be determined. LASSO regression seems to be well suited for optimizing the chromosomal regulatory sequence, which lie on a a probability simplex, since a probability simplex always has a 1-norm of 1.

2.4 Shaping Edit Chromosomes

Shaping is a technique that has been developed by Ashlock and Ashlock (2014) and McEachern and Ashlock (2014). Shaping a genetic algorithm is done by putting restrictions or conditions on how the algorithm transitions from one generation to the next, or how each generation is tested. Ashlock and Ashlock (2014) looked at how different shapes yield different results for the agents that are forced by an evolutionary algorithm to play an iterated prisoners dilemma. McEachern and Ashlock (2014) showed that shaping a finite stated machine could increase the performance of said finite state device when acting upon a supervised classification task. The results of this thesis lead to an independent rediscovery of shaping.

2.5 Conclusion

Graph colouring is an NP-hard problem that has been applied in a variety of ways since

the 1850's. Since the 1970s, classical and modern graph colouring algorithms have been developed to find proper k-colours, where k is close to the chromatic number of the graph, for more and more complex graphs for many different applications. As graphs have been used for more applications, harder graphs have been discovered. In order to deal with these harder graphs, more advanced modern graph colouring algorithms have been developed. As graph colouring algorithms have been developed, finding hard to colour graphs, for a particular graph colouring algorithm, has become a harder problem to solve.

Evolutionary algorithms have been used to help optimize search algorithms. As modern graph colouring algorithms become more complex, they have been able to colour increasingly difficult graphs. This thesis will look into whether evolutionary algorithmic techniques could help improve these modern graph colouring algorithms, by trying to create a technique that is able to find graphs that are hard for these graph colouring algorithms to colour. This thesis looks at whether a genetic algorithm, developed by Ashlock *et al.* (2014) could be used to find harder to colour graphs. Since the genetic algorithm used is rather complex, the parameter optimization is rather important. Discerning an effective way to optimize the parameters of the genetic algorithm is the main focus of these thesis. This thesis will analyse a new method of optimizing the parameters of genetic algorithms, comparing its effectiveness to particle swarm optimization and differential evolution. As genetic algorithms continue to be developed and implemented, different meta-heuristic optimization solutions will continue to be developed and evolved. This is done with the hope of decreasing the computational load while increasing or maintaining the performance of the optimization of the genetic algorithms.

Chapter 3

Methodology

As previously explained, the increase in computational power, allowed evolutionary computation to develop, allowing genetic algorithms to become practical in the mid 1980s. Timmins and Ashlock (2017) have showed that genetic algorithms can be used to shape graphs for different applications when studying epidemics. It has been shown that evolutionary algorithms can be used in graph colouring, though they are outperformed by other types of heuristic solvers (Fleurent C. and Ferland J.A., 1996). This thesis will look at a genetic evolutionary algorithm that has been created specifically to find hard to colour graphs for a given graph colouring algorithm. All genetic algorithms use parameters to create solutions, but as Eiben *et al.* (1999) explain, the genetic algorithm is only as good as its parameter settings. There have been many different techniques developed that optimize the parameters of a genetic algorithm; however, as the genetic algorithms become more complicated, these different techniques become too computationally intense. Genetic algorithms have more parameters than just the chromosomal regulatory sequence. These parameters are chosen before optimization techniques are used to find good chromosomal regulatory sequences.

This thesis compares a number of optimization techniques on the parameters in an evolutionary algorithm in an attempt to discern in which situations certain parameter optimization techniques are useful. It compares the effectiveness of the classical parameter optimization techniques – particle swarm optimization and differential evolution – to a new technique called beam optimization. This new technique is very specialized to the type of genetic algorithm being used, specifically looking at the operators involved in the genetic algorithm and how they effect how the evolutionary algorithm searches the solution space. The specific genetic algorithm being optimized is based on the genetic evolutionary algorithm, used by Ashlock, Schonfeld, Barlow and Lee (2014), that has been adapted in an attempt to try to

find hard to colour graphs given a graph colouring algorithm. The parameter optimization technique is so specialized to the operators that are being used in the specific genetic algorithm, the results are only useful for this specific genetic algorithm. If this technique is going to be used to optimize other genetic algorithms, a similar analysis must be done on the parameters of the other genetic algorithms.

3.1 The Genetic Evolutionary Algorithm

The genetic algorithm used by Timmins and Ashlock (2017) is a complicated one, with a large number a ways to change the way it performs. Despite this complexity the nine parts of a genetic algorithm are clearly defined. Understanding each of these aspects of the genetic evolutionary algorithm is key in analysing its performance, and trying to optimize the parameters of the genetic algorithm. In looking at these nine different parts it is clear which parameters need to be set before starting to optimize the chromosomal regulatory sequence and which parameters are in the chromosomal regulatory sequence.

When using evolutionary algorithms, the specific problem that is imposed on the algorithm is typically not the same as the problem that is being solved. It is typical that the fitness function will relate the possible solutions to the problem that is imposed on the evolutionary algorithm, to the problem that is actually being analysed. The problem is to find hard to colour graphs, given a graph colouring algorithm, but this is not the problem imposed on the genetic algorithm. The problem that is imposed on the genetic algorithm is to find the edit chromosome, that when applied to the original graph, produces the hardest to colour graph for the given graph colouring algorithm (1).

For this evolutionary algorithm, the original graph is a difference graph of order 64 (64 vertices) with a difference set $\{\pm 1, \pm 2\}$. This graph is generated by placing the vertices in a ring and then connecting each vertex in the graph to the two closest vertices on both the right and left sides of the vertex. This graph is used because it is vertex-transitive, simple to encode and appears to have the appropriate density (Ashlock *et al.*, 2014). The graph colouring algorithm used in this thesis is the classical RS method.

The solution set, or population members, for the genetic algorithm used in this thesis, are edit chromosomes. These edit chromosomes are composed of seven edit commands(A-G), seen in Figure 3.1. These edit commands can be viewed as the nucleotides of the edit chromosome. As these edit chromosomes are "read", each of the edit commands are applied the initial graph, creating new graphs (2). The seven edit commands are as follows:

Add (A) adds an edge between two vertices. If an edge already exists, the add operator does nothing.

Delete (B) removes an edge between two vertices. If no edge exists, if does nothing.

Toggle (C), adds an edge between two vertices, if there does not already exist an edge between them and removes an edge, if there is an existing edge between the two vertices.

Local Toggle (D), is similar to toggle, but acts locally. This mutation takes a vertex A, walks to an adjacent vertex B. It then walks to a vertex C, which is adjacent to B. Local toggle then toggles the connection between A and C.

Hop (E) is applied to a subset of three vertices. If there is a path connecting vertex A and C via vertex B, but A and C are not adjacent, hop will remove one of the existing edges in the path connecting A and C, and add an edge to connect vertices A and C directly.

Swap (F) is applied to a subset of four vertices. This subset of vertices must have an edge connecting vertices A and B as well as an edge connecting vertices, C and D. There must be no edges connecting either A or B to C or to D. Swap will remove the edges connecting A to B and C to D and will add an edge connecting A to C or D and B to the remaining vertex.

Null(G), will do nothing. This edit command allows for variation in the length of the edit chromosome being applied since it is an operator that acts as if no operator is being applied.

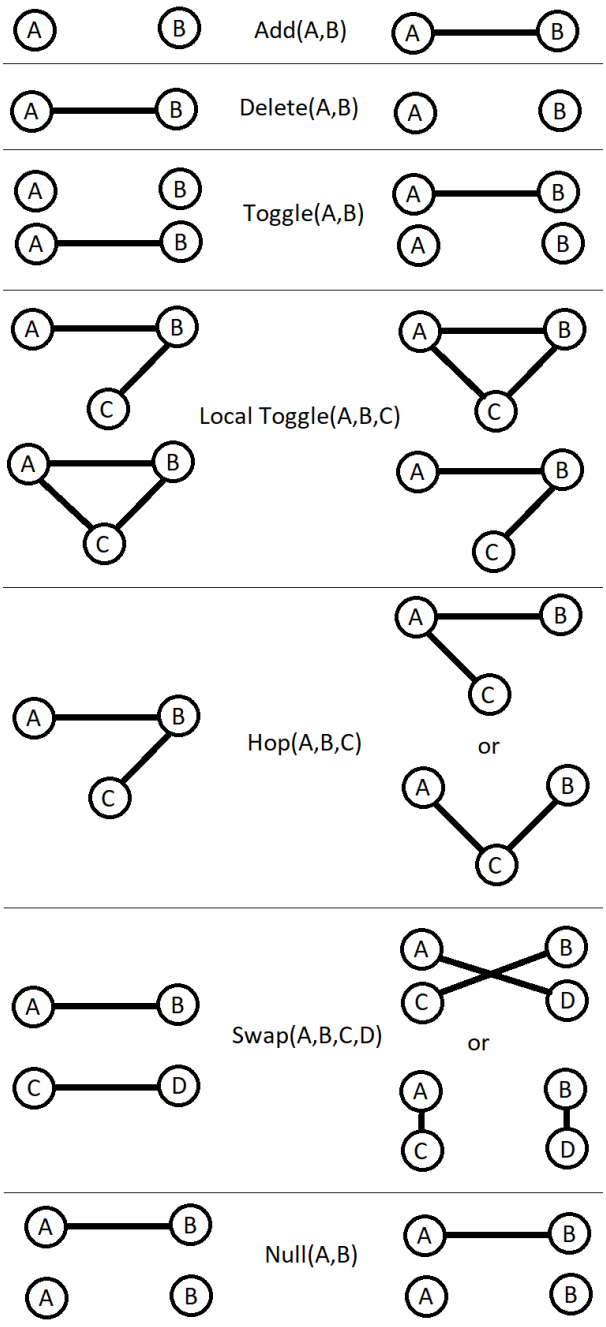


Figure 3.1: Different Edit Commands

Since the chromosomal regulatory sequence regulates the likelihood of each edit command

that is used, it is defined by a linear combination of all seven edit commands, $(a_1, a_2, a_3, a_4, a_5, a_6, a_7)$ such that $\sum_{i=1}^{7} a_i = 1$ and $a_i \geq 0$: where a_i, $i = 1, 2, ..., 7$ represent the likelihood of toggle, hop, add, delete, swap, local toggle, and null respectively. This thesis specifically looks at different methods used to try and find at what likelihood each of edit commands, in the chromosomal regulatory sequence, should be set at so the genetic algorithm can perform optimally. In other words, this thesis attempts to find the optimal chromosomal regulatory sequence for the genetic algorithm.

The initial population of edit chromosomes is generated using the chromosomal regulatory sequence, in such a way that a bit slicing technique can be used to select the edit command and parameters needed by the edit commands. Each of the edit commands need to select a number of vertices and their neighbours. Add, delete, and toggle need two vertices. Local toggle and hop need a vertex and two neighbours. Swap needs two vertices and two neighbours. Null needs does not need any vertices or neighbours selected. The edit commands need at most two vertices selected and two neighbours selected. The two vertices and neighbours are selected by randomly selecting two large numbers using the command

$$LN = lrand48()\%1000000$$

Once the two large numbers have been selected. The mod(64) of this large number is taken to determine the vertex selected.

$$vertex = LN mod(64)$$

Once the vertex has been selected the large number is divided by 64, using integer division (64 is used in both cases, since the graph is of order 64).

$$LN = LN/64$$

The mod(degree of the vertex selected, $d(vertex)$) of the large number is taken to determine which neighbour is selected.

$$neighbour = LN mod(d(vertex))$$

When the edit command is applied it only uses the information that is needs. If the edit command only needs one vertex it takes the first one selected. If it does not need any

neighbours selected, it ignores the neighbours selected. The two large numbers for each edit chromosome are stored in pairs in a chromosomes.

The chromosomal regulatory sequence is used to determine which edit command is being used when. To determine what edit command is being used a number C is uniformly randomly selected from the interval [0,1]. Once C has been selected, the likelihood of toggle a_1 is subtracted from C to create C_1. If C_1 is less than 0, toggle is the edit command chosen. Otherwise a_2 is subtracted from C_1 to create C_2. If C_2 is selected hop is the edit command. This continues until the edit command has been determined. Once the edit command it must be stored with the information about which vertices and neighbours are selected.

The first number, of the two large numbers, is modified to contain the information for which edit command is being used, without changing the number. This is done by multiplying the number by 7 (the number of edit commands) and then adding the edit command number (cn) (0 for toggle, 1 for hop, 2 for add, 3 for delete, 4 for swap, 5 for local toggle and 6 for null).

$$LN = 7 * LN + cn$$

The edit command can the be retrieved by taking $LN mod(7)$ and the original large number can be retrieved by taking $LN/7$, using integer division. Once the first of the two large numbers has been modified, the two large numbers are stored in the sequence and are treated as one edit command. When the edit command is applied to the original graph, it applies each of the edit commands in the order of the chromosome to the specific vertices defined by the large numbers stored in the edit chromosome. Even though the edit commands are stored using two large integers, they can be thought, and evolve, as if they were a single edit command (Figure 3.2).

In order to create the initial population of edit chromosomes, the chromosomal regulatory sequence is used to build a predetermined number of edit chromosomes (3). As the evolutionary algorithm evolves, the population size for every generation remains the same size as the initial generation (4). The population size, the predetermined number of chromosomes, is one of the first parameters that needs to be set before one can analyse the performance of the different edit chromosomes optimize the chromosomal regulatory sequence.

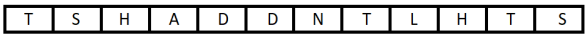


Figure 3.2: A Possible Edit Chromosome of length 12

Each edit chromosome that is made must be tested to determine how hard it is for the

graph colouring algorithm to colour the graph that is made when the edit chromosome is applied to the original graph. This is done, just like in all evolutionary algorithms, with an objective function. This objective function, also known as a fitness function, must determine how hard to colour the graphs generated are, for the graph colouring algorithm used by the genetic algorithm. Despite the fact that the definitions for slightly-hard-to-colour graphs and hard-to-colour graphs are very discrete definitions, the degree to which a graph is hard to colour, for a particular graph colouring algorithm, lies on a continuum. When different graph colouring algorithms are applied to a graph a multiple number of times, the number of colours used to colour this graph may vary. This is because of the stochastic nature of graph colouring algorithms. Two graphs that are both defined as hard-to-colour for a graph colouring algorithm probably differ in hardness. The graph colouring algorithm may find a proper colouring that uses a number of colours that is close to the chromatic number of the first graph. This same graph colouring algorithm may not be able to find a proper colouring of another graph that is close to the chromatic number of the second graph. Both graphs would be considered hard to colour, but the second graph would be considered much harder to colour that the first graph.

The fitness function used by the genetic algorithm is designed to try and determine out where on this continuum of hardness to colour a particular graph lies. This fitness function has the graph, generated by the edit chromosome, coloured 50 times, by the given graph colouring algorithm. The number of colours used in each colouring of the graph is then used to determine the fitness of the graph. The fitness function takes the average number of colours used by the different colourings and divides it my the minimal number of colours used by all colourings. This is the fitness of each member of the population. Even though a common way to analyse the effectiveness of a graph colouring algorithm for a particular graph is to use a performance guarantee, which determines the fitness by setting it equal to the largest number of colours used, by all of the colourings, to colour the graph, this thesis does not use a performance guarantee.

A performance guarantee is effectively measures how badly a graph colouring algorithm can perform on a graph. Determining the worst possible outcome is an interesting characteristic, but when determining how hard a graph is to colour, this may not be useful. When trying to colour a graph, the largest number of colours used may be much larger than the chromatic number of the graph. If the graph colouring algorithm rarely has a colouring that uses a number of colours much larger than the chromatic number, while normally colouring the graph with a number of colours that is much closer to the chromatic number of the

graph, this graph would be easier to colour than a graph that is consistently over-coloured by the same graph colouring algorithm.

The fitness function used by this genetic algorithm theoretically will increase when the graph that is coloured is harder to colour for the graph colouring algorithm.

$$Fitness = \frac{Average\ Number\ of\ Colours\ Used\ to\ colour\ the\ graph}{Minimal\ Number\ of\ Colours\ Used\ to\ colour\ the\ graph} \quad (5)$$

A weakness of this fitness function is that the minimal number of colours used to colour the graph may not be an accurate representation of the actual chromatic number of the graph. Since we are trying to find a high ratio between the lowest number of colours used and the average number of colours used, this is not a concern. If the ratio is large, the graph will be hard to colour. As the graphs become harder, more of the colourings will require a larger number of colours, while the number colourings that use a small number of colours will decrease. It is important to note this fitness function does not determine if a graph is hard-to-colour or not. Even if all the graphs that are being compared are hard to colour for the graph colouring algorithm, this fitness function can still be used to estimate how hard these graphs are compared to each other.

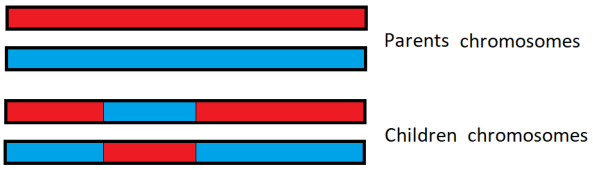


Figure 3.3: Two Point Crossover

Now that the objective function has been defined, the iterative portion of the genetic algorithm can be defined, starting with the selection scheme. The selection scheme (6) is to randomly select five of the members of the population to be the parents for the next generation. To create the new populations (7), the two most fit individual edit chromosomes of the five are duplicated. The two duplicate edit chromosomes will have a two point crossover mutation (Figure 3.3), and then five random point mutations to the edit commands (Figure 3.4). When a point mutation occurs, one edit command is randomly replaced using the same process that was used to create the initial population. The likelihood of each different type of edit command occur-

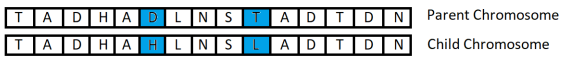


Figure 3.4: Two Point Mutations

ring is still governed by the same chromosomal regulatory sequence. Note that it could be replaced with the edit command that was already there. Once these two new edit chromosomes are created they replace the worst two of the five edit chromosomes that were selected. These five edit chromosomes then replace the five edit chromosomes that were taken from the initial population to create the new generation. The fitness of the edit chromosomes are then tested by applying two new edit chromosomes to the original graph and colouring the resulting graphs using the graph colouring algorithm. Once its fitness has been determined, it is then integrated into the next generation of the population (8). The next parent generation is then generated by randomly selecting five new parents, in the same way. This cycle continues until a fixed number of iterations have been applied. The genetic algorithm is more likely to find an optimal solution with a larger the number of iterations, but it is also more computationally intense. This will differ for different applications of the genetic algorithm. In this thesis, when the genetic algorithm is run with a given parameter setup, it runs 30 times, each time with the same parameter setup. The output is the fitness of the best edit chromosome from each of the 30 runs. When different parameters of the genetic algorithm are being optimized the performance of the genetic algorithm is the average of the fitness of the best edit chromosome from each of the 30 runs.

Eiben *et al.* (1999) show that since genetic evolutionary algorithms are so reliant on their parameters in order for the genetic algorithm to preform well, an optimal chromosomal regulatory sequence must be determined. In order to simplify parameter optimization, all of the parameters that are not edit commands, in the chromosomal regulatory sequence, will be set before trying to optimize the chromosomal regulatory sequence. These parameters are the population size, the mutation number, and the number of iterations.

3.2 Setting the Population Size, Mutation Number and Iteration Number

In order to set the population size, mutation number and number of iterations of the genetic algorithm, the genetic algorithm must be run. In order to run the genetic algorithm, a chromosomal regulatory sequence must be used. This chromosomal regulatory sequence could be any chromosomal regulatory sequence. It is better to optimize the population size, mutation number and number of iteration using a chromosomal regulatory sequence that contains each of the edit commands. Because of this, a chromosomal regulatory sequence

where each edit command has the same likelihood is used. In determining the appropriate settings for these parameters, performance has to be balanced with computational load. The goal of setting these parameters is to find an appropriate balance between computational load and performance.

In order to set these, 45 experimental simulations were done. There were three types of runs based on computational load; a fast run, which used 100,000 generations in the genetic algorithm; a medium speed run, which used 1,000,000 generations: and a slow run, which used 10,000,000 generations. Each of these runs has 15 different experiments. There were five different population sizes used, each with three different point mutation numbers. The population sizes were: 10 chromosomes; 32 chromosomes; 100 chromosomes; 320 chromosomes; and 1000 chromosomes. The three different numbers of point mutations were 1, 3 and 5 (Table 3.1). An increase in the number of iterations increases the computational load,

		10 000 Iterations			100 000 Iterations			1 000 000 Iterations		
		Number of Point Muations			Number of Point Muations			Number of Point Muations		
		1	3	5	1	3	5	1	3	5
Population Size	10	Results			Results			Results		
	32									
	100									
	320									
	1000									

Table 3.1: Experiments used to set Population Size, Mutation Number and Iteration Number

while the mutation number and population size only modifies how the genetic algorithm runs. An increase in computational load is only desirable if the results significantly improve. The results for each of the experiments is the performance of the genetic algorithm run with the parameter settings.

The best performing parameter setting, mutation number and population size, from each of the runs, is compared to the other runs to determine if the increase in the number of iterations was worthwhile. The appropriate number of iterations is decided by determining what the appropriate balance between computational load and performance. Once the number of iterations is chosen, the best performing parameter setting from that run of tests is chosen. Once the population size, number of mutations, and number of iterations has been set, the chromosomal regulatory sequence can be tested to try to determine the optimal chromosomal regulatory sequence.

3.3 Testing the Edit Commands of Chromosomal Regulatory Sequence in Genetic Algorithm

Knowing effects that the edit commands have on how the genetic algorithm searches the solution space may shed some light on how the genetic algorithm should be optimized. In order to try and discover how the edit commands affect how the genetic algorithm searches the solution space, each of the edit commands was tested. All of the edit commands, except the null operator were tested individually, including an experiment against the null operator. The add and delete operators were tested together to try and determine how the characteristics of the other edit commands change as the graphs change in size (number of edges).

In order to test the effect that toggle, hop, add, delete, swap and local toggle each have on how the evolutionary algorithm searches the solution space, two runs of 10 groups of experiments were conducted for each edit command. In both runs of experiments each of the 10 groups of experiments were created using a fixed ratio of the 5 edit commands that are not being test and the edit command that is being tested. In the second run of experiments the null operator is also used. In the first run of experiments, in each group of 11 experiments the chromosomal regulatory sequence is made by using the edit command that is being tested and the fixed ratio of the other edit commands. The amount of the chromosomal regulatory sequence, that is to say the likelihood of the edit command that is represented by the edit command being tested, ranges from 0 percent to 30 percent, differing by 3 percent in each experiment. The rest of the chromosomal regulatory sequence was made up of the appropriate amount of the other five edit commands according to the fixed ratio for the given group of experiments. For the second run of tests, the fixed ratio of a group of tests represents 70 percent of the chromosomal regulatory sequence. The remaining portion of the chromosomal regulatory sequence was made up of the edit command being tested and the null operator. The amount of this remaining portion of the chromosomal regulatory sequence that was made up of the edit command that was being tested ranges from 0 to 30 percent, differing by 3 percent in each experiment. Whatever portion of the chromosomal regulatory sequence that was not represented by the 6 edit commands was represented by the null operator. Table 3.2 shows an example of a possible group that could be used to test swap in the first run and the second run.

In addition to all of these experiments, which are designed in an attempt to understand the tendencies of each operator, while interacting with the other operators, each edit

	Toggle	Hop	Add	Delete	Swap	Local Toggle	Null	
Fixed Ratio	0.15	0.1	0.3	0.05		0.1		
Run 1 Group 1								
Test	Toggle	Hop	Add	Delete	Swap	Local Toggle	Null	Results
1	0.214286	0.142857	0.428571	0.071429	0	0.14285714	0	
2	0.207857	0.138571	0.415714	0.069286	0.03	0.13857143	0	
3	0.201429	0.134286	0.402857	0.067143	0.06	0.13428571	0	
4	0.195	0.13	0.39	0.065	0.09	0.13	0	Results
5	0.188571	0.125714	0.377143	0.062857	0.12	0.12571429	0	
6	0.182143	0.121429	0.364286	0.060714	0.15	0.12142857	0	
7	0.175714	0.117143	0.351429	0.058571	0.18	0.11714286	0	
8	0.169286	0.112857	0.338571	0.056429	0.21	0.11285714	0	
9	0.162857	0.108571	0.325714	0.054286	0.24	0.10857143	0	
10	0.156429	0.104286	0.312857	0.052143	0.27	0.10428571	0	
11	0.15	0.1	0.3	0.05	0.3	0.1	0	
Run 2 Group 1								
Test	Toggle	Hop	Add	Delete	Swap	Local Toggle	Null	Results
1	0.15	0.1	0.3	0.05	0	0.1	0.3	
2	0.15	0.1	0.3	0.05	0.03	0.1	0.27	
3	0.15	0.1	0.3	0.05	0.06	0.1	0.24	
4	0.15	0.1	0.3	0.05	0.09	0.1	0.21	
5	0.15	0.1	0.3	0.05	0.12	0.1	0.18	Results
6	0.15	0.1	0.3	0.05	0.15	0.1	0.15	
7	0.15	0.1	0.3	0.05	0.18	0.1	0.12	
8	0.15	0.1	0.3	0.05	0.21	0.1	0.09	
9	0.15	0.1	0.3	0.05	0.24	0.1	0.06	
10	0.15	0.1	0.3	0.05	0.27	0.1	0.03	
11	0.15	0.1	0.3	0.05	0.3	0.1	0	

Table 3.2: Testing Swap

command was isolated and studied individually, using the null operator to vary the representation of each operator. Eleven experiments were conducted on each operator, varying the representation of each edit command from 0 to 100 percent, in 10 percent increments. Table 3.3 shows the testing of Toggle with Null.

In order to test the effect the ratio of the add and delete operators have on how the genetic algorithm searches the solution space, 10 groups of 11 experiments were conducted. In each of the group, the add and delete edit commands combined to represent 30 percent

Test	Toggle	Hop	Add	Delete	Swap	Local Toggle	Null	Results
1	0	0	0	0	0	0	1	
2	0.1	0	0	0	0	0	0.9	
3	0.2	0	0	0	0	0	0.8	
4	0.3	0	0	0	0	0	0.7	Results
5	0.4	0	0	0	0	0	0.6	
6	0.5	0	0	0	0	0	0.5	
7	0.6	0	0	0	0	0	0.4	
8	0.7	0	0	0	0	0	0.3	
9	0.8	0	0	0	0	0	0.2	
10	0.9	0	0	0	0	0	0.1	
11	1	0	0	0	0	0	0	

Table 3.3: Testing Toggle with just Null

of the chromosomal regulatory sequence, while the other 70 percent was made up of a fixed ratio of the other edit commands. The add and delete edit commands range from 0 percent add and 30 percent delete to 0 percent delete and 30 percent add, incremented by 3 percent between each experiment. The fixed ratio of the other 5 parameters were created by manually randomly selecting the ratio of the parameters. Once one fixed ratio was randomly chosen the next fixed ratio was randomly chosen, with the bias to be different from the previously chosen ratios. This way the add and delete edit commands are analysed when used with a variety of chromosomal regulatory sequences, which each emphasize a different edit command. Table 3.4 is an example of a possible group that could be selected.

These experiments that isolate individual parameters were designed to determine the qualitative effects each edit command has on how the genetic algorithm searches the solution space. To determine how effective the genetic algorithm can be, the chromosomal regulatory sequence must first be optimized. Three different parameter optimization techniques have been analysed in the thesis in an attempt to determine in what situations the different techniques should be used. This may give light to which parameter optimization technique should be used to find the chromosomal regulatory sequence that should be used to find the edit command that when applied to the difference graph creates graphs that are relatively hard to colour for the given graph colouring algorithm.

Test	Toggle	Hop	Add	Delete	Swap	Local Toggle	Null	Fitness
1	0.2	0.1	0	0.3	0.05	0.25	0.1	
2	0.2	0.1	0.03	0.27	0.05	0.25	0.1	
3	0.2	0.1	0.06	0.24	0.05	0.25	0.1	
4	0.2	0.1	0.09	0.21	0.05	0.25	0.1	Results
5	0.2	0.1	0.12	0.18	0.05	0.25	0.1	
6	0.2	0.1	0.15	0.15	0.05	0.25	0.1	
7	0.2	0.1	0.18	0.12	0.05	0.25	0.1	
8	0.2	0.1	0.21	0.09	0.05	0.25	0.1	
9	0.2	0.1	0.24	0.06	0.05	0.25	0.1	
10	0.2	0.1	0.27	0.03	0.05	0.25	0.1	
11	0.2	0.1	0.3	0	0.05	0.25	0.1	

Table 3.4: Testing Add and Delete

3.4 Optimizing the Edit Commands of the Chromosomal Regulatory Sequence

As stated earlier, genetic algorithms were originally believed to be robust enough that as long as the parameter settings were balanced, the genetic algorithm would compensate for sub-optimal parameter settings (Goldberg, 1989). By 1999, Eiben *et al.* were able to explain why parameter optimization is necessary when using genetic algorithms. In an attempt to optimize the chromosomal regulatory sequence of the genetic algorithm, three techniques were used. The first two techniques, particle swarm optimization and differential evolution, are traditional methods that have been adjusted so they can be used on a probability simplex. The third method, which used the characteristics of the edit commands to optimize the edit commands, is a new technique is introduced with the aim to decreases the computational load required to optimize the chromosomal regulatory sequence of the genetic algorithm.

In order to optimize the chromosomal regulatory sequence, there is a fitness function that evaluates the performance of the chromosomal regulatory sequence. The performance of the genetic algorithm is used as the fitness function for the different chromosomal regulatory sequences that are being used. All of the parameter optimization methods scan the solution space of the chromosomal regulatory sequence in an attempt to find the fittest chromosomal regulatory sequence.

3.4.1 Particle Swarm Optimization

Particle Swarm Optimization is the oldest parameter optimization method that is being used in this thesis. The particle swarm optimization technique that is used is a modified version of the particle swarm optimization technique that was tested by Shi and Eberhart (1998). It is made of four parts: initializing the population; evaluating the current population; evolving the population; and ending the particle swarm optimization. Each member of the population is known as a particle. As stated earlier, particle swarm optimization is usually used to analyse a finite region of a non-bounded continuous space. Since the particle swarm optimization is being used to optimize the chromosomal regulatory sequence, adjustments are made to the initialization and evolutionary portions of the particle swarm optimization.

Three different versions of particle swarm optimization were used, differing only in how the current population evolves. Computational constraints, when evaluating the current population, limits the number of particles and the number of iteration in the particle swarm optimization.

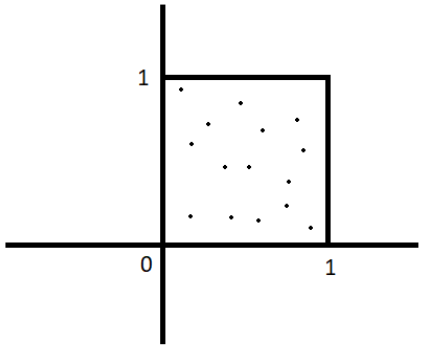


Figure 3.5: Uniform Sampling

Initialization

For all particle swarm optimizations, the first variable that is set is the population size. The particles, members of the population, represent different chromosomal regulatory sequences. These particles are tested in parallel and then analysed. Do to computational constraints, a population size of 20 was used. In order to initialize the location of each particle for the particle swarm optimization, the particles are uniformly randomly selected from the probability simplex. Usually this defined region is simple enough that in order to take a uniform random sample, one would take a uniform random sample over each of the variables, as scene in Figure 3.5. Since the particles represent different possible chromosomal regulatory sequences, they must lie on a probability simple. The sum of the likelihoods of each of the seven edit commands is equal to one, and all of the likelihood of each of the possible edit commands are non-negative. $(a_1, a_2, a_3, a_4, a_5, a_6, a_7)$ such that $\sum_{i=1}^{7} a_i = 1$ and $a_i \geq 0$.

Performing a uniform random sampling over a probability simplex requires some care. Simply taking a uniform random sample of each variable from 0 to 1, and then normalizing the vector, using the 1 norm, does not provide a uniform random sample. This is easiest shown on a two dimensional probability simplex. If we take a uniform random sample of each variable from 0 to 1, the likelihood of each point on the square in Figure 3.10 will be the same. This means the likelihood a point will be randomly selected from each of the triangles is equal to the area of the triangle, since that area of the square is equal to 1. When the lines are normalized, using the 1 norm, the points in triangles will be mapped to the line in each triangle. As seen in Figure 3.6, the length of the normalized lines in each of the triangles are approximately the same. In order for the random sample of the simplex to possibly be uniform, the area of each of the triangle needs to be the same. As seen in Figure 3.6, the area of the triangles are very different. The area of the yellow and blue triangles are both 1/6 units squared and the area of the red and green triangles are both 1/3 units squared (see Appendix A for solution). The vector that is created by taking a uniform random sample of each of the two parameters and then normalizing the resulting vector is twice as likely to be in the green and red triangles than it is to be in the blue and yellow triangles. This clearly shows that this method of randomly sampling will not provide a uniform random sampling on our simplex.

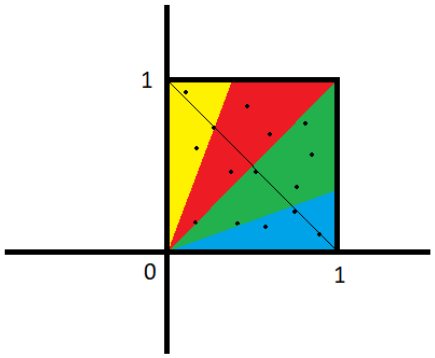


Figure 3.6: Uniform Sampling

As explained by Willms (2018) one can achieve a uniform random sample of any simplex by first taking a uniform random sample over one of the variable and then applying a conversion function to the sampling. This process is continued until all of variables have been selected. The conversion function is as follows.

$$x_i = P_{n-i}\left(f_i, 1 - \sum_{j=1}^{i-1} x_j\right), \quad \text{where} \quad P_n(f,h) = \begin{cases} h[1-(1-f)^{1/n}] & \text{if } n \geq 1, \\ h & \text{if } n = 0. \end{cases}$$

The process Willms has described requires that the solution space must be a simplex that satisfies $\{x \in \mathbb{R}^n | \sum_{i=1}^{n} x_i = 1, x_i \geq 0\}$. All of the possible chromosomal regulatory sequences used by the genetic algorithm lie on a 7-dimensional standard simplex, meaning $n = 7$. This

process of uniformly random sampling is used to select the initial population of particles.

Normally an initial velocity is also randomly generated for the particle in the particle swarm optimization. If a random velocity is used, the particle will probably leave the simplex. In ensure that the particles do not leave the simplex, an initial velocity of zero is used. An initial velocity that would ensure that the particles do not leave the simplex, could have been made for each of the particles by simply taking another uniform random sample and taking the difference between the new particle and the original particle. For this thesis, an original velocity of zero was used.

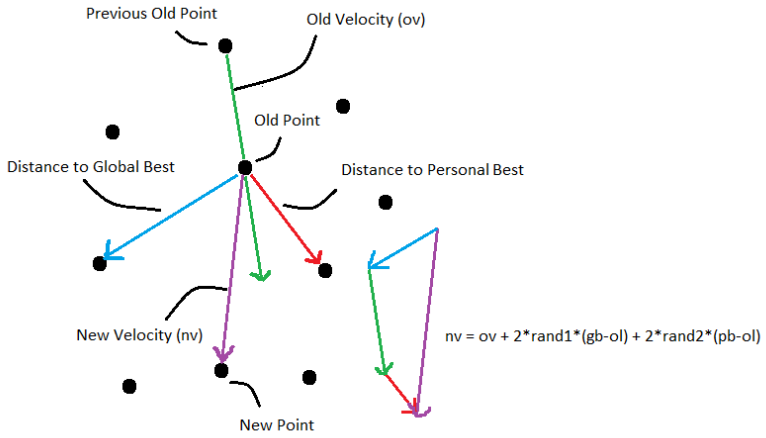


Figure 3.7: Evolution of a Particle

Evaluating the Population

For every generation of the population, the location of each of the particles is a possible chromosomal regulatory sequence. Each of the particles' performances are determined by running the genetic algorithm using each particle's location as the chromosomal regulatory sequence. When a new generation is created, it is then evaluated. In order to increase the speed of this evaluation, every particle in a generation is tested in parallel.

Evolving the Population

Particle swarm optimizations update the locations (nl) of each particle by adding a velocity vector (vv) to the old location (ol) of each of the particles.

$$nl = ol + vv$$

Similar to Shi and Eberhart's particle swarm optimization (1997), the new velocity vector that is added (vv) is based on the old velocity of the particle (ov), the particle's current location (ol), the particle's best location (pbl) and the overall best location of all the particles

(gbl).

$$vv = ov + 2 * c1 * (pbl - ol) + 2 * c2 * (gbl - ol)$$

$c1$ and $c2$ are chosen by taking a uniform random sample of the interval [0,1] (Figure 3.7). The difference between the three different types of evolutionary algorithms is what happens if the new location of the particle is outside of the bounds of the probability simplex. In other words, how will the particle swarm optimization evolve if the new location of the particle involves a negative likelihood for one or more of the edit commands.

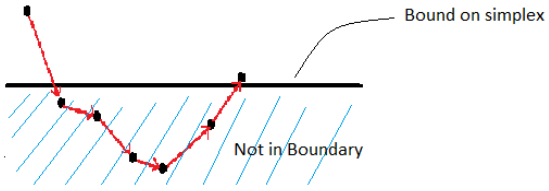
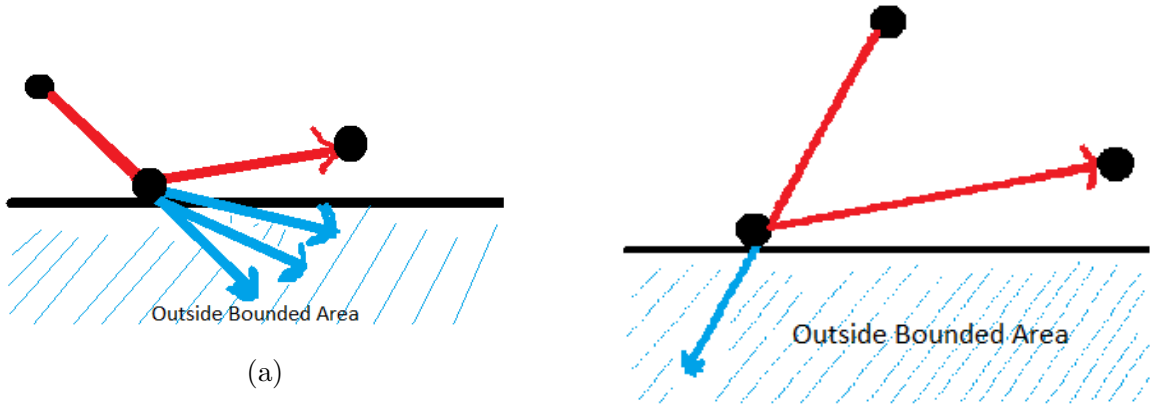


Figure 3.8: Evolution of First Particle Swarm Optimization

The first particle swarm optimization was run the genetic algorithm using the invalid location as the chromosomal regulatory sequence, trusting that in future generations the particle will be brought back, or almost back into the valid region on the simplex (Figure 3.8). It is important to note that the evolutionary algorithm will still run with negative numbers in the chromosomal regulatory sequence, but the likelihood of each edit command will not be the same as the location of the particle, i.e. the particle will not accurately represent the chromosomal regulatory sequence that is being tested. This version of particle swarm optimization is dangerous because it could end up with a solution that is not in the bounds of the simplex. If this is the case, the results would have to be interpreted to discern the actual chromosomal regulatory sequence that has been found.

The second particle swarm optimization uses the boundary as a stopping point. If a particle is going to move past one or more boundaries causing the likelihood of one or more of the edit commands to become negative, this particle swarm optimization would allow the particle to move in that direction until it hits the first boundary, at which point it would stop. The velocity of the particle which is used to generate the next generation's velocity is not changed. This particle would be "stuck" on this boundary until the new velocity no longer has the particle moving outside of the that particular boundary (Figure 3.9(a)). This method allows the particle to maintain its directional momentum, but also stay within the boundaries of the simplex. One of the issues with this method is that if the global best location and the personal best location of a particle are both on the same boundary as the current particle, its velocity will never direct the particle back into the simplex. This means

that the particle can "get stuck" on a boundary. The other issue is that the velocity may change so much, before it is directed back into the middle of the simplex that the particle just jumps from boundary to boundary without exploring the middle of the simplex.



(a) (b)

Figure 3.9: Evolution of the Second (a) and Third (b) Particle Swarm Optimizations

The third particle swarm optimization also uses the boundary of the simplex in a similar way to the second particle swarm optimization. If the velocity vector were to take the particle outside the boundary, the particle moves in the direction of the velocity vector until it hits the first boundary it comes in contact with. The particle then hits that wall, and its velocity is eliminated; similar to what happens when a Velcro tennis ball hits at Velcro wall, its velocity is eliminated. The "old velocity" that is used for the next iteration has a velocity of zero (Figure 3.9(b)). The advantage to having a zero velocity is that the particle will never be stuck at a point on a boundary, unless its personal best and the global best points are also at that point on the boundary. The disadvantage is that the particle loses its momentum and can now move in any direction, based solely on the particle's best location and the global best location of all of the particles. This limits the area of the probability simplex that is explored by these particles.

Ending the Particle Swarm Optimization

The particle swarm optimization that Shi and Eberhart (1998) used ended after 4000 iterations, and it featured 20 particles. The particle swarm optimization was seen as a failure if it did not find an acceptable solution within the 4000 iterations. Because of the computational load of the genetic algorithm being used, the three particle swarm optimizations

that are tested use 100 iterations. It is important to remember that the goal of testing the three different types of parameter optimization methods is to try and find optimal chromosomal regulatory sequences in a timely matter. Even with the low iteration number, and the small number of particles, the three particle swarm optimizations take about 17 days each. The main drawback about using particle swarm optimization to determine the parameters of genetic algorithms is that as genetic algorithms become more complex, particle swarm optimization will take more time to optimize them.

3.4.2 Differential Evolution

Differential Evolution scans the parameter solution space very differently than Particle Swarm Optimization. Like Particle Swarm Optimization, Differential Evolution has four main parts. The four parts of differential evolution are: initializing and evaluating the initial population; creating and evaluating the mutant generation; selecting the new generation; and ending the differential evolution (Storn and Prince, 1997). For differential evolution the members of each generation are called agents.

Just like particle swarm optimization, differential evolution is usually designed to scan a finite region of a infinite continuous space. There are three types of differential evolution being used, which differ only in the creation of the mutation generation. They are both modified to be adapted to the bounded simplex. The most substantial change, made in all three differential evolutions, is that there is no recombination when creating the mutant generation.

Initialization and Evaluation of the Initial Agents in Differential Evolution

Just like in particle swarm optimization, differential evolution initializes its population using a uniform random sample of the solution space. Since the agents represent the chromosomal regulatory sequences, which lie on a probability simplex, the same technique, described by Willms (2018), that was used to initialize the particles in particle swarm optimization is used to initialize the agents in the differential evolution. Twenty agents are initialized for the initial population. Storn (2013) suggests that the number of agents used should be 10 times the number of parameters that are being optimized. It is limited to 20 to decrease computational intensity. The agents are normally stored as a chromosome, where each nucleotide is a location of an agent with its corresponding fitness. To avoid confusion, because the idea of a chromosome is used in the genetic algorithm, the agents will be stored in a

chain, where each link in the chain will store an agents location and performance (Figure 3.10).

Figure 3.10: Initial Population Chain

The performance of the agents in the differential evolution is evaluated the same way as the particles in the particle swarm optimization were evaluated. The genetic algorithm is run with the location of the agent being the chromosomal regulatory sequence. The fitness of the genetic algorithm, when run using the agent's location as the chromosomal regulatory sequence, is the performance of the agent.

Creating the Mutant Population

Each of the three differential evolutions will create a mutant chain that has a mutant agent for every agent in the original chain in the generation (Figure 3.11).

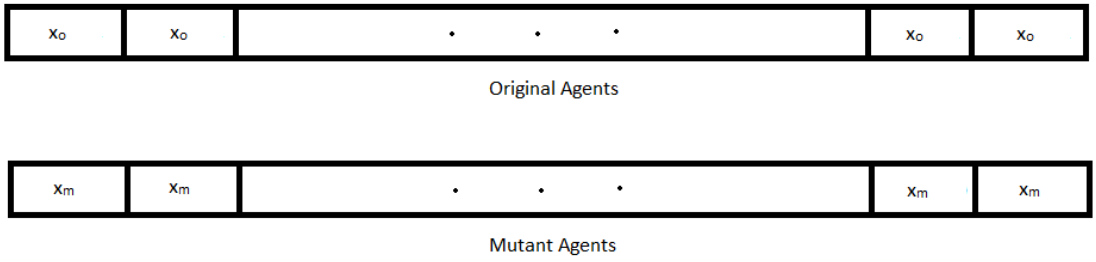


Figure 3.11: Mutant Chain and Original Chain

The first differential evolution that is used does not have any restrictions when it comes to the boundary of the simplex. There is no recombination during the mutation stage of the differential evolution. As shown before, each agent has a specific location in the chromosome. For each agent (x_o), there is at mutant agent (x_m) in a corresponding chain where

$$x_m = a + F * (b - c).$$

the variables a, b and c are all distinct agents from the original chain of that generation (Figure 3.12(a)). They are randomly chosen, but must not equal each other or x_o. F is the selective weighting factor. The mutant agent's performance is then evaluated by running the genetic algorithm with the location of the agent as the chromosomal regulatory sequence. F is a selective weighting factor that is randomly selected from the interval $[0.5, 1.0]$, because Storn (2013) suggests that this helps the differential evolution when the updating mechanism has a noisy evaluation, which probably is the case here since the genetic algorithm's fitness is noisy.

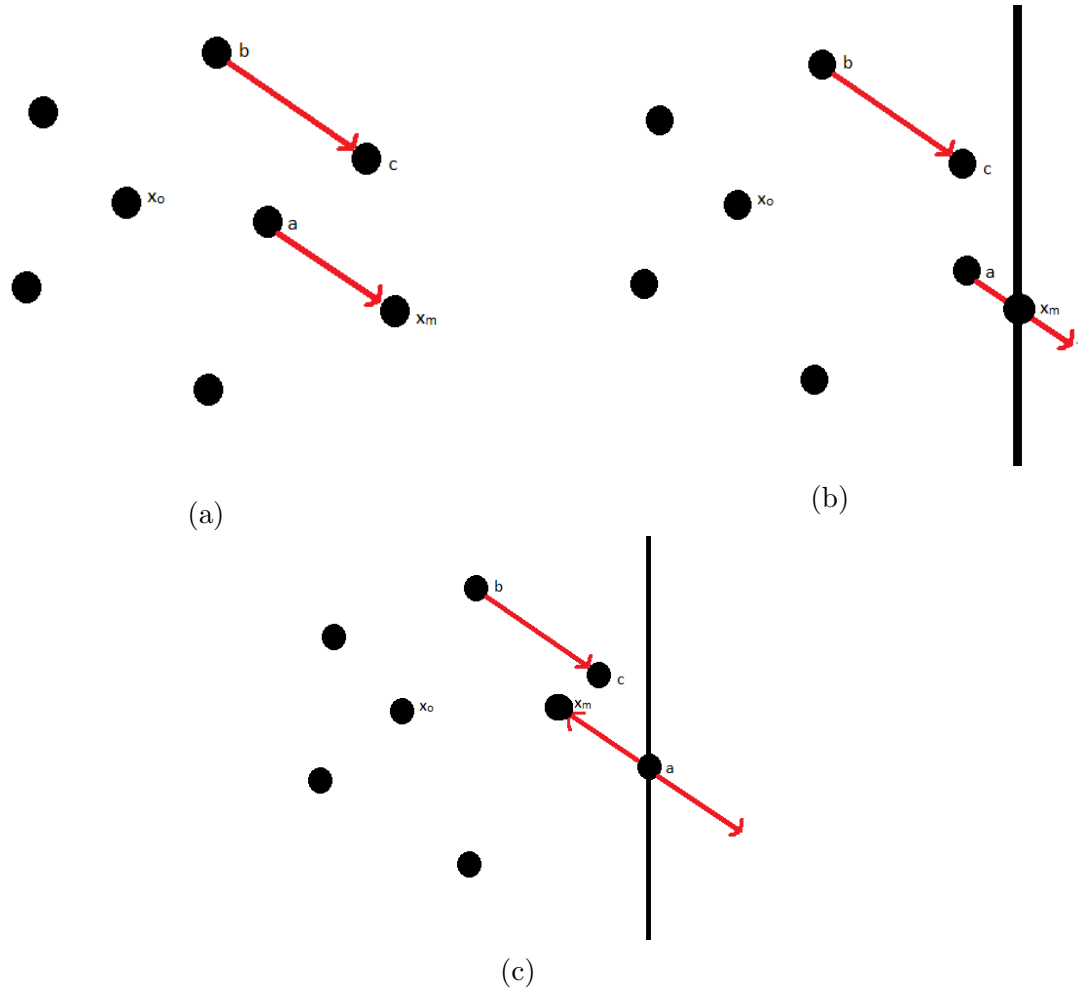


Figure 3.12: Generating the Mutant Agents for the First (a), Second (b), and Third (c) Differential Evolution

The second differential evolution uses the same equation to generate the mutant agents

as the first differential evolution.

$$x_m = a + F * (b - c)$$

However, it limits the location of the agents to the boundary of the probability simplex. If the vector $F * (b - c)$, when added to a, causes the likelihood of one of the edit commands in the chromosomal regulatory sequence, represented by x_m, to become negative, an alternative F, F_o, is used. F_o causes x_m to travels as far as it can alone the vector $(b - c)$ without, crossing the boundary of the probability simplex. If agent a is already on the boundary, the mutant agent is a (Figure 3.12(b)). The mutant agent's performance is then determined using in the same was as the first differential evolution.

The third differential evolution is very similar to the second differential evolution. The only difference is that if agent a is located on a boundary and $F_o = 0$, then another selective weighting factor, $F_{o1} = -F$, is used. If F_1 brings a beyond another boundary, the selective weighting factor F_2 that brings a along the vector $(c - b)$ until it reaches the first boundary, is used. If this selective weighting factor is also zero, the mutant agent's location is a (Figure 3.12(c)). Just like in the first and second differential evolution, the mutant agents fitness is the fitness of the genetic algorithm that has uses the location of the agent as the chromosomal regulatory sequence.

Selecting the new generation

Once the mutant chain is created, the original chain and the mutant chain are compared. If the fitness of the link of the mutant chain is equal to or better than the fitness of the corresponding link in the original chain, it is selected as the link for the next generation's chain, otherwise the original agent is selected (Figure 3.13).

Ending the Differential Evolution

The differential evolution is run for 40 iterations to limit the computational intensity. Normally differential evolution is run for thousands of iterations. Even with the low number of agents and the limited number of iterations, the differential evolution takes over 6 days.

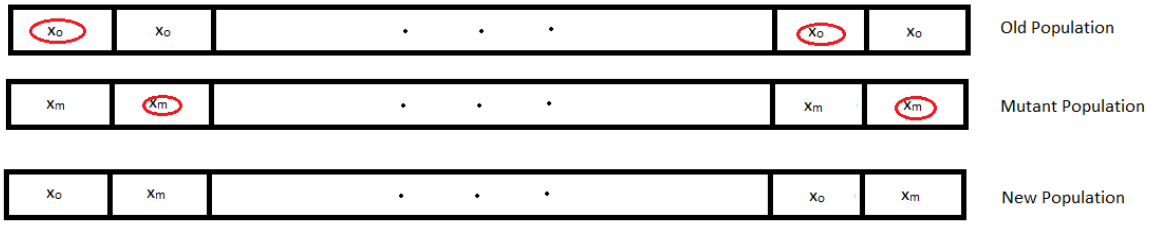


Figure 3.13: Selecting the New Generation of Agents

3.4.3 Optimizing the Chromosomal Regulatory Sequence using Beam Optimization

Beam optimization uses the characteristics of each of the edit commands, to try and optimize the chromosomal regulatory sequence in a specific order. Branches represent different combinations of edit commands. Each branch is tested to determine which branch should be used to continue the search. Eiben (1997) has made the argument that the randomness of the genetic algorithm creates too much noise to be able to discern the impact that each of the parameters in the genetic algorithm have. As genetic algorithms become more complex and the parameters become more drastically different, like the edit commands used in this genetic algorithm, this appears to no longer be true. It may be possible to discern some of the characteristics that different edit commands have on how the genetic algorithm searches the solution space. Ashlock *et al.* (2014) have shown that the swap operator removes complexities in graphs and tends to create graphs that have low eccentricity.

This method for optimizing the chromosomal regulatory sequence uses the results from the experiments that were designed to determine the effects that the different edit commands have on the genetic algorithm. The main characteristic that is used by beam optimization is the fitness that the chromosomal regulatory sequence tends toward.

The results of the experiments described earlier to test the impact of various edit commands, are used to order the edit commands. It is important to note that the null operator is not being tested as an operator, it is being used to provide variability in the length of the edit chromosomes used by the genetic algorithm. As such, the null operator is seen, where necessary, to have be the best performing operator of any of the operators. Once this has been done, the optimization of the chromosomal regulatory sequence can begin. This

technique uses the order of the edit commands in two different ways. The first method is called optimal ratios, and the second method is called optimal individuals.

Beam Optimization using Ratios

The first method, beam optimization using optimal ratios, compares two edit commands (A and B) at a time to find the optimal ratio of the two edit commands. This new ratio is made into a new "edit command" replacing the previous two edit commands. For simplicity, the ratio will be expressed in such a way that the sum of the two terms is 100. In order to determine the optimal ratio of two edit commands, a series of two runs are performed. The first run of experiments contains 11 experiments. These experiments have chromosomal regulatory sequences that range from 0 percent of the first edit command (A) to 100 percent of the first command, where each experiment increases the likelihood of edit command A by 10 percent each time. The remaining portion of the chromosomal regulatory sequence is made up of the second edit command (B), in every experiment. These experiments determine the best branch down which to proceed searching. Just like in the particle swarm optimization and differential evolution, the performance of each of the chromosomal regulatory sequences is determined by its fitness.

The next run of experiments is a set of 21 experiments based around the results of the 11 experiments from the first run. These 21 experiments are the 21 chromosomal regulatory sequences that are centred around the best performing chromosomal regulatory sequence from the first run of experiments. For example, if the best result from the previous run of experiments is 70 percent the first command and 20 percent the other. The next run of experiments would range from 60 to 80 percent of the first edit command and 40 to 20 percent of the other edit command. If the best chromosomal regulatory sequence is 100 percent of one of the two edit commands and zero percent of the other edit command, the 21 experiments are centred around 90 percent of the one edit command, and 10 percent of the other edit command. Each of the experiment differ by 1 percent. Once these 21 chromosomal regulatory sequences have been tested using the genetic algorithm, the chromosomal regulatory sequence with the best fitness the new "edit command".

The common ratio of the seven edit commands is determined by first determining the best common ratio of the two edit commands that seem to have create graphs with the lowest fitness to create a new "edit command" with that tends toward the lowest fitness (Figure 3.14). Once this has been done, the best ratio between this "edit command" and the next least performing edit command, of the remaining edit commands, is determined. This

Part One				Part Two			
	A	B			A	B	
1	0	100		1	30	70	
2	10	90		2	31	69	
3	20	80		3	32	68	
4	30	70		4	33	67	
5	40	60	<-- Winner	5	34	66	
6	50	50		6	35	65	
7	60	40		7	36	64	
8	70	30		8	37	63	<-- Winner
9	80	20		9	38	62	
10	90	10		10	39	61	
11	100	0		11	40	60	
				12	41	59	
				13	42	58	
				14	43	57	
				15	44	56	
				16	45	55	
				17	46	54	
				18	47	53	
				19	48	52	
				20	49	51	
				21	50	50	

(a) Individual A and B are replaced by an individual that is 37 percent A and 67 percent B.

Part One				Part Two			
	A	B			A	B	
1	0	100		1	80	20	
2	10	90		2	81	19	
3	20	80		3	82	18	
4	30	70		4	83	17	
5	40	60		5	84	16	
6	50	50		6	85	15	
7	60	40		7	86	14	
8	70	30		8	87	13	
9	80	20		9	88	12	
10	90	10		10	89	11	
11	100	0	<-- Winner	11	90	10	
				12	91	9	
				13	92	8	
				14	93	7	<-- Winner
				15	94	6	
				16	95	5	
				17	96	4	
				18	97	3	
				19	98	2	
				20	99	1	
				21	100	0	

(b) Individual A and B are then replaced by an individual that is 93 percent A and 7 percent B.

Figure 3.14: Different Possible Solutions for Linear Combinations of the Individuals A and B (percentages).

is done repeatedly until there is only one "edit command" left. Let the order of the edit commands, when they are ordered from worst performing edit command to least performing edit command, be A, B, C, D, E, F, G. Let a new "edit command" determined by the optimal ratio of two edit commands be denoted by (AB), and let $\Rightarrow$ represent running the two runs of experiments to determine the optimal ratio of two edit commands. The common ration of the seven edit commands would be determined in the following order. Null is seen as having the smallest step size and is tested last.

$$A, B, C, D, E, F, G \Rightarrow (AB), C, D, E, F, G \Rightarrow ((AB)C), D, E, F, G \Rightarrow (((AB)C)D), E, F, G$$

$$\Rightarrow ((((AB)C)D)E), F, G \Rightarrow (((((AB)C)D)E)F), G \Rightarrow ((((((AB)C)D)E)F)G)$$

Figure 3.15 shows the progression of the optimization of the chromosomal regulatory sequence using Optimal Ratios. The "edit command" $((((((AB)C)D)E)F)G)$, shown by the last line of Figure 3.16 would be seen as the optimal chromosomal regulatory sequence for the genetic algorithm. Null is seen as performing the best at creating hard to colour graphs and is represented by white in Figure 3.15.

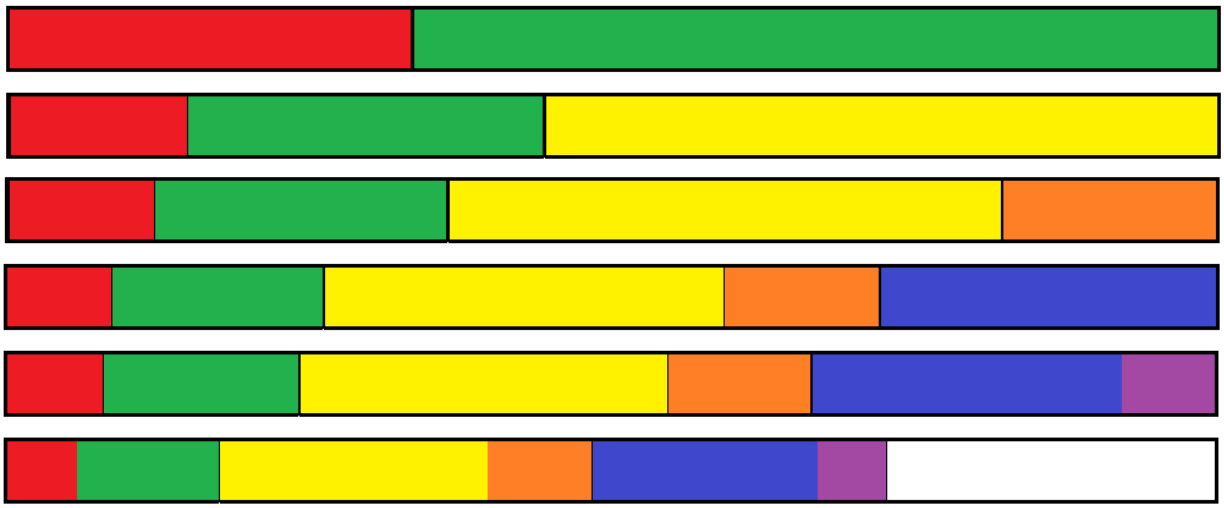


Figure 3.15: Beam Optimization using Optimal Ratios

Beam Optimization using Optimal Individuals

This method was motivated by an interesting result from testing the different edit commands against the null operator, which is discussed further in the results chapter. The optimal chromosomal regulatory sequence that uses one edit command and the null operator is not 100 percent the edit command and zero percent the null parameter. This method systematically optimizes the amount of each of the edit commands that should be used in the chromosomal regulatory sequence.

The structure of Optimal Individuals is very similar to Optimal Ratios. It uses two runs of experiment, where the first run has 11 evenly distributed experiments followed by the second run, which has 21 evenly distributed experiments centred around the best chromosomal regulatory sequence from the first 11 experiments. Optimal Individuals optimizes the amount of each edit command, starting from the edit command that seems to create the graphs that are easiest to colour to the edit command that seem to make the hardest. The main difference is that the amount of each of the individual edit commands that should be used in the chromosomal regulatory sequence is determined using the null parameter and the edit command itself. It does not change any of the other edit commands being used.

The first two runs of experiments in Optimal Individuals are very similar to the first two runs of experiments in Optimal Ratios. The first run 11 experiments use chromosomal regulatory sequences that range from 0 percent of the edit command that creates the easiest

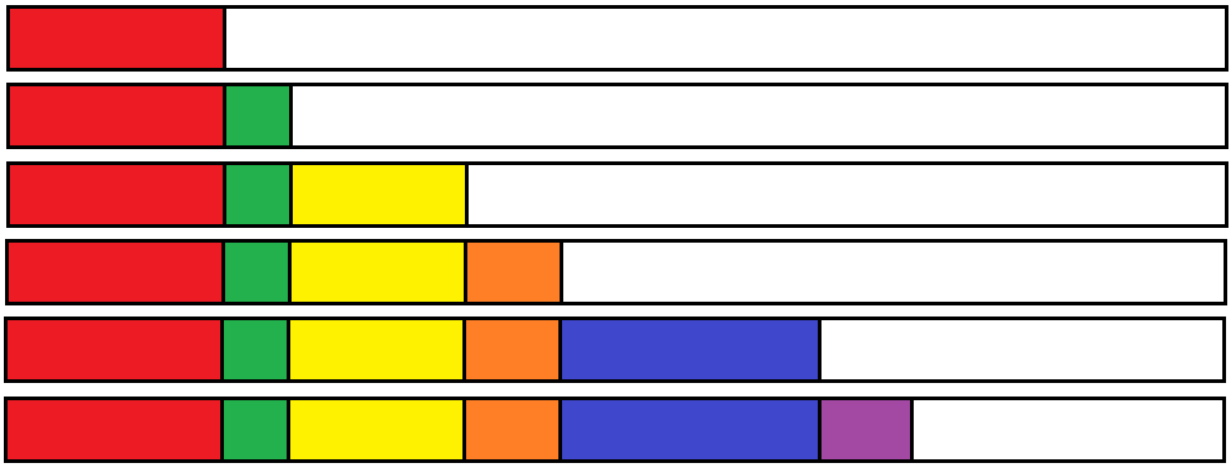


Figure 3.16: Optimizing a Chromosomal Regulatory Sequence using Optimal Individuals

to colour graph (A) to 100 percent. The remainder of the chromosomal regulatory sequence is made up of the null operator. The next run of experiments, uses 21 experiments that are centred around the chromosomal regulatory sequence with the best performance. The main difference is that the remainder of the chromosomal regulatory sequence is made up of the null edit command. Once the optimal amount of any particular edit command has been determined, that amount of the edit command is fixed. When the optimal amount of one edit command is determined, the edit command that seems to create graphs that are the next easiest to colour, is determined.

When determined the optimal individual amount of each of the remaining edit commands, the chromosomal regulatory sequence is not split into 100 equal parts. The portion of the chromosomal regulatory sequence which is not made up of the edit commands that have already been determined is split up into 100 equal parts, and then optimized in a similar fashion to the first step of the chromosomal regulatory sequence. Once every one of the edit commands have been optimized in this order, excluding the null edit command, the final chromosomal regulatory sequence is seen to be the most optimal chromosomal regulatory sequence.

Figure 3.16 is shows the evolution of Optimal Individuals. Each bar is the result of optimizing the amount of each edit command, with the last line being the final optimized chromosomal regulatory sequence. The white space represents the null operator

3.5 Conclusion

When using computationally intense programs, like finding an optimal chromosomal regulatory sequence of this genetic algorithm, one must find the appropriate balance between the duration of a program and the program's performance. In order to optimize the chromosomal regulatory sequence of this genetic algorithm, first the parameters that are not in the chromosomal regulatory sequence are optimized. Then the chromosomal regulatory sequence itself can be optimized.

The parameters that are not in the chromosomal regulatory sequence, specifically Population Size, Mutation Number and Iteration Number, are optimized in an attempt to find the settings that perform well enough to determine the different between difference chromosomal regulatory sequences, while not being too computationally intense.

The parameters in the chromosomal regulatory sequence are then optimized. First, each of the edit commands are tested in an attempt to determine general characteristics of each of the parameters. Then four different techniques were used to try and optimize the chromosomal regulatory sequence: Particle Swarm Optimization; Differential Evolution; Optimal Ratios; and Optimal Individuals. Particle Swarm Optimization and Differential Evolution are both more computationally intense, but limit the restriction of the solution space that is searched. Optimal Ratios and Optimal Individuals both have less computational load, but constrict the section of the solution space which is scanned. The results of these experiments may give light to where each of these methods of parameter optimization techniques should be used, based on the computational load and performance of the different parameter optimization techniques (Figure 3.17).

Chapter 4

Results

The genetic algorithm has been tested three different ways for three different purposes. First, the parameters that are not in the chromosomal regulatory sequence have been optimized to increase the performance of the evolutionary algorithm. Then, each of the edit commands in the chromosomal regulatory sequence were individually tested to determine their general characteristics. Finally, the chromosomal regulatory sequence was optimized using multiple different techniques do determine which technique should be used to optimize the parameter settings when evolutionary algorithms become significantly complex. All three ways the genetic algorithm was tested were designed to specifically analyse particular parts of the genetic algorithm.

The first set of experiments determines how the genetic algorithm operates and determines what setup is useful for testing and optimizing the chromosomal regulatory sequence. The second set of experiments determine the general characteristics of the edit commands and in which ways they could be used in the future. The last set of experiments determines which type of parameter optimization technique should be used on the genetic algorithm, given specific time constraints. All of the experiments shed light on how the genetic algorithm could be used in the future, specifically for finding harder to colour graphs for different graph colouring algorithms.

4.1 Optimizing the Parameters of the Genetic Algorithm that Are Not in the Chromosomal Regulatory Sequence

The goal of optimizing the parameters that are not in the chromosomal regulatory sequence is to find a parameter setting that has significantly different outcomes for different chromosomal regulatory sequences, while still not being too computationally intense that it would be too hard to optimize the chromosomal regulatory sequence. The main way to control the computational load is to control the iteration number of the genetic algorithm, which determines the number of generations in the genetic algorithm. The mutation number and the population size modify how the population changes overtime.

The fast run of experiments (Table 4.1), which uses 100,000 generations in the genetic algorithm, does not give any strong conclusions. The run time is about 3 minutes. The best run has a population size of 320 and 1 point mutation per evolution. There are two general trends when the genetic algorithm uses 100,000 iterations. The larger the population size is the better the genetic algorithm performs, and a genetic algorithm with a smaller number of point mutations performs better than larger number of point mutations. Experiment 13 - 15 however, go against these general trends. The results of all of these experiments have little difference between them. Two explanations for this can be: the different settings have very little impact on the performance of the genetic algorithm; or the genetic algorithm does not have enough generations to allow the genetic algorithm to evolve sufficiently to show the differences between the parameter settings. The experiments that used 1 000 000 generations and 10 000 000 generations give light to whether there is not enough generations in the population or if the non chromosomal regulatory sequence parameters do not have much of an impact on how the genetic algorithm

Experiment #	Population	# of Mutations	Fitness Rate Average
1	1000	1	2.08489
2	1000	3	2.050033
3	1000	5	1.987955567
4	320	1	2.110778
5	320	3	2.036556
6	320	5	2.046611667
7	100	1	2.019168333
8	100	3	1.975778333
9	100	5	2.010723
10	32	1	1.864333
11	32	3	1.838789
12	32	5	1.854443667
13	10	1	1.723499667
14	10	3	1.7810551
15	10	5	2.0525

Table 4.1: Results of the Short Run testing the Non-Chromosomal Regulatory Sequence Parameters (100 000 generations)

runs.

The medium run of experiments (Table 4.2), which used 1 000 000 generations in the genetic algorithm, has results that are much clearer than the results from the run of experiments which used 100 000 generations. The run time is about 30 minutes. The trend that a larger population size performs better than a small population size as well as the trend that a larger number of point mutations is more effective than a smaller number of point mutations are both supported with a larger disparity between the fitness of the genetic algorithm in each experiment. The trend that the a larger number of point mutations is more effective that a smaller number of point mutations is the opposite trend that was seen in the Short run. In this run of experiments a population of 1000 is much more effective than the other population sizes. Apart from experiments that use population sizes of 10 and 32, the performances of genetic algorithms are significantly better in the medium run than the short run, specifically in the large population sizes. An increase in the fitness of about 0.15-0.25 justifies the increase of computational load.

Test	Population	MNM	Average Fitness #
1	1000	1	2.222555667
2	1000	3	2.2351112
3	1000	5	2.235554667
4	320	1	2.181722333
5	320	3	2.184445
6	320	5	2.195999333
7	100	1	1.968444333
8	100	3	2.074332667
9	100	5	2.028223
10	32	1	1.822166667
11	32	3	1.9457
12	32	5	1.959388333
13	10	1	1.789511333
14	10	3	1.960722333
15	10	5	2.059110333

Table 4.2: Results of the Medium Run testing the Non-Chromosomal Regulatory Sequence Parameters (1 000 000 generations)

The long run of experiments (Table 4.3), which use 10 000 000 generations, in the genetic algorithm takes about four days to complete. The Medium run of tests has ten times the number of iterations at the first run of tests and takes ten times longer to run. This suggests that the evolutionary algorithm takes the most time during the evolutionary portion of the genetic algorithm. The long run has ten times the number of iterations as the medium run but takes much longer than ten times as long to run. The reason why the genetic algorithm may have taken so much longer for the long run of experiments, is that the genetic algorithm takes enough space on the computer to slow it down. Just like in the previous experiments, the experiments with a larger population performed better and the experiments with a larger number of mutations tend to perform better. In these experiments however, there is very little difference between the performance of the algorithm with a point mutation number of

5 and one with a point mutation number of 3, particularly when the population is larger. This suggests that if given enough time, the cross-over mutation will compensate for the lack of point mutations, showing the robustness of genetic algorithms. All three experiments show that a large population performs better than a small population. Unlike the increase in the number of generations from 100 000 to one million, the increase in performance of the genetic algorithm, when increasing the population size from 1 000 000 to 10 000 000, is not significant enough to merit the increase in computational load. The fitness increases significantly for the parameter settings with a smaller population. These smaller populations are still out-performed by the genetic algorithms with the larger populations, which have an increase in fitness of about 0.0666 which does not merit the large increase in computational load.

The settings for testing the genetic algorithm, are chosen based on the results from the 45 experiments. The results with the highest fitness from the short run of experiments had a average fitness of 2.085, with an standard deviation of 0.1989. The results with the highest fitness from the medium run of experiments had a fitness of 2.236, with a standard deviation of 0.0576. The results with the highest fitness from the longest run of experiments had a fitness of 2.321, with a standard deviation of 0.1280. The settings from the medium run were chosen (1 000 000 generations, a mutation number of 5, and a population size of 1000). This was chosen, not only because the mean increased, but also because the standard deviation is much smaller. When using the genetic algorithm to find the hardest to colour graphs for a given graph colouring algorithm, the chromosomal regulatory sequence must be optimized. A small standard deviation makes the changes in the performance of the algorithm much more significant. Even-though the long run out-performs the medium and short runs where not chosen because the small increase in fitness. Once an optimal chromosomal regulatory sequence is found, one can increase the population size to find better edit

Test	Population	MNM	Average Fitness #
1	1000	1	**2.291333667**
2	1000	3	**2.302000333**
3	1000	5	**2.303555**
4	320	1	**2.250055333**
5	320	3	**2.216333333**
6	320	5	**2.216333333**
7	100	1	**2.117444667**
8	100	3	**2.159111**
9	100	5	**2.196221333**
10	32	1	**2.002666667**
11	32	3	**2.026111333**
12	32	5	**2.120667667**
13	10	1	**1.952222**
14	10	3	**2.024**
15	10	5	**2.167944667**

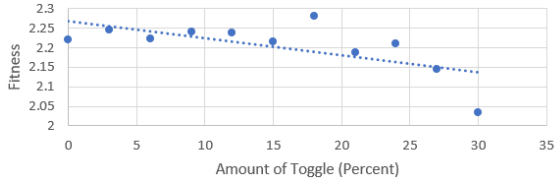
Table 4.3: Results of the Long Run testing the Non-Chromosomal Regulatory Sequence Parameters (10 000 000 generations)

chromosomes that create hard to colour graphs for the given graph colouring algorithm.

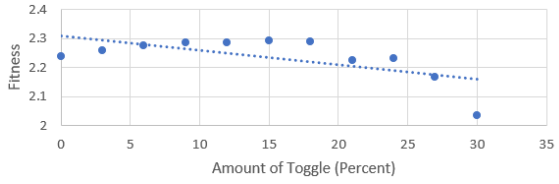
4.2 The General Characteristics of Each of the Edit Commands in the Chromosomal Regulatory Sequence and How They Impact the Size of the Graphs

By looking at how changing the linear combination of the likelihoods of the edit commands in the chromosomal regulatory sequence impacts the genetic algorithm's effectiveness, insights on the general characteristics of each type of edit command can be seen. Two distinct characteristics are looked at. First, all of the edit commands are tested individually to determine individual characteristics, then they were tested using add and delete to determine how these commands impact the size (number of edges) of the graph. Appendix B has the graphs of the results of the experiments used to determine the characteristics of the individual edit commands. To make the analysis of the data computationally easier, the performance of the genetic algorithm is determined by the sum of all the fitness scores, instead of the average.

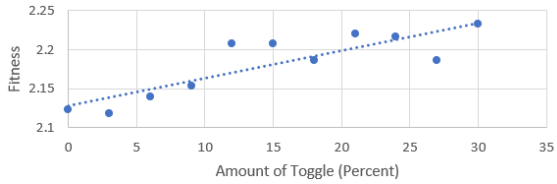
There are two interesting results: an increase in each edit command seems to create either harder to colour graphs or easier to colour graphs; and it appears that a linear function does a resonable job at fitting the data. The optimal fitness that the edit commands tend toward, do not always tend to the same side of the graph. Sometimes less of a particle edit command is better, while other times, more of the edit command is better. This seems to be more related to the absolute amount of the edit command being tested, rather than how it compares, relative to the other edit commands. The second run of experiments shows more linearity that the first run of experiments. This further suggests that some sort of linear relation may exist between variables.



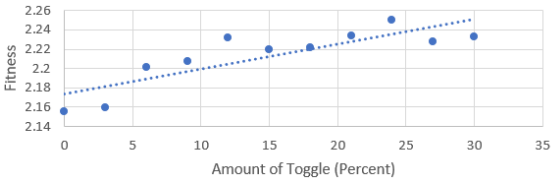
(a) Run 1 Group 1



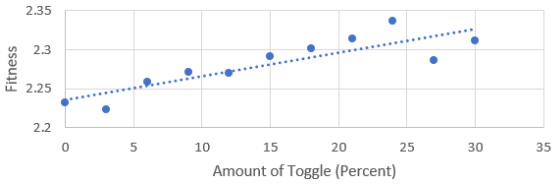
(b) Run 2 Group 1



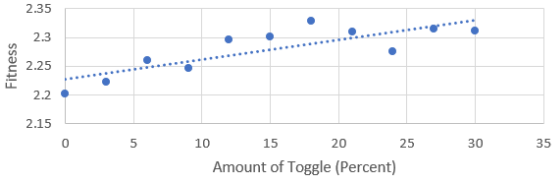
(c) Run 1 Group 2



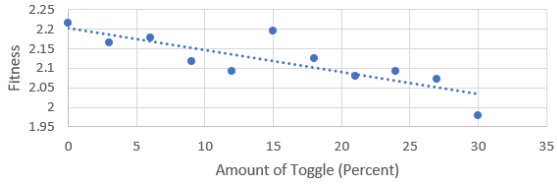
(d) Run 2 Group 2



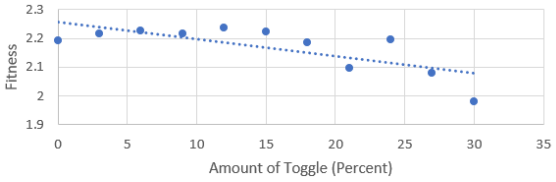
(e) Run 1 Group 3



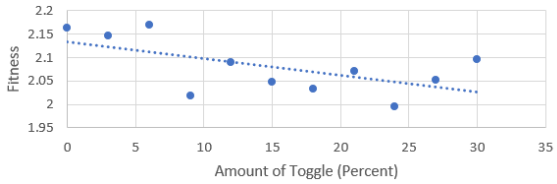
(f) Run 2 Group 3



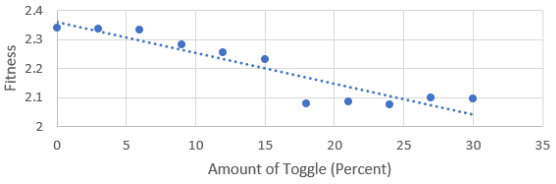
(g) Run 1 Group 4



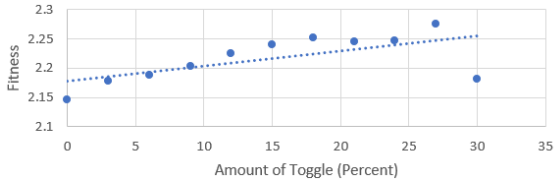
(h) Run 2 Group 4



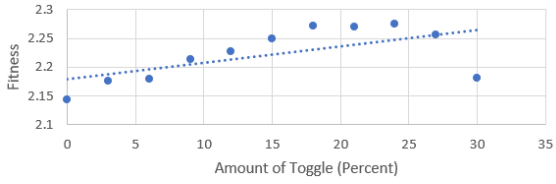
(i) Run 1 Group 5



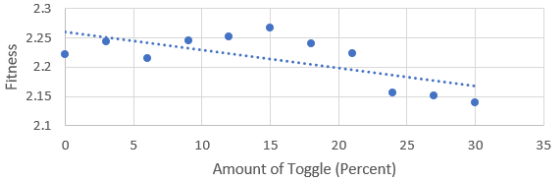
(j) Run 2 Group 5



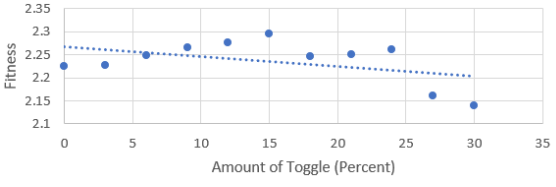
(k) Run 1 Group 6



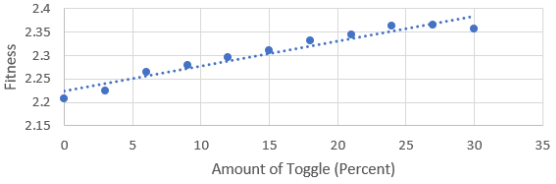
(l) Run 2 Group 6



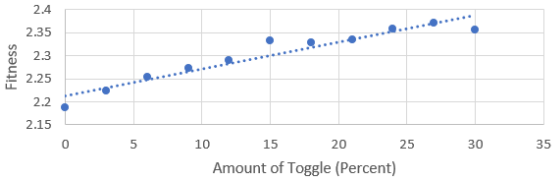
(m) Run 1 Group 7



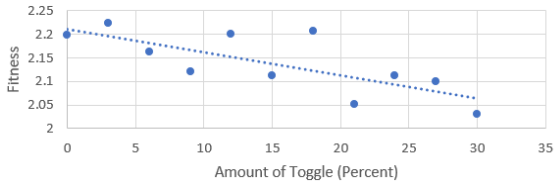
(n) Run 2 Group 7



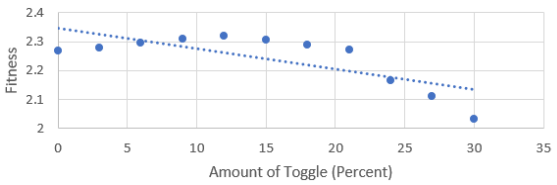
(o) Run 1 Group 8



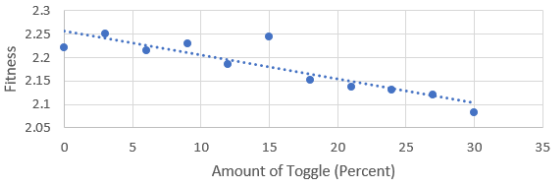
(p) Run 2 Group 8



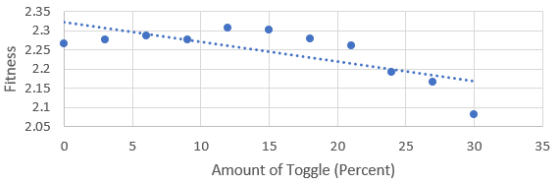
(q) Run 1 Group 9



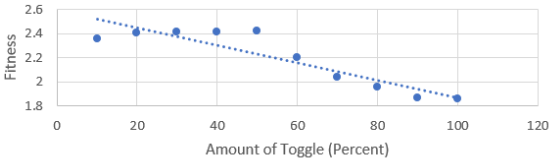
(r) Run 2 Group 9



(s) Run 1 Group 10



(t) Run 2 Group 10

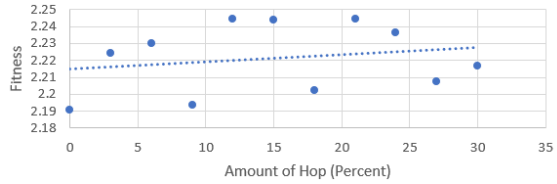


(u) Null Experiment

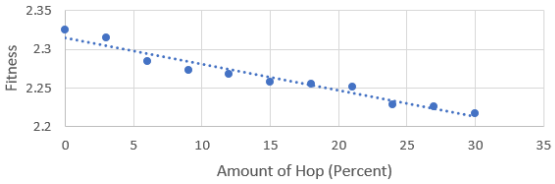
Figure 4.1: Toggle Experiments

The results from the experiments that test Toggle can be seen in Figure 4.1. The best fitness values for Toggle are between 2.3 and 2.367, when the other parameters are set appropriately. When there is more add and less delete, less toggle tends to be more effective; while, when there is more delete and less add, more toggle tends to be more effective. This is an interesting result since, it is known that harder graphs tend to have a smaller size. If toggle adds about the same number of edges as it removed, one would expect that the amount the of add and delete edit commands present would have a smaller impact than it does. This suggests that toggle either tends to add more edges that it removes or vice-versa. If toggle tended to remove more edges than it added, then it would perform better with more delete than add, but would still perform better with more toggle if there were more add than delete. If toggle tended to add more edges than it removed, then it would perform worse with more add that it would with more delete. If there was a lot of add, more toggle would probably decrease the performance of the genetic algorithm. The more delete, the better toggle would perform. If the amount of delete was significantly larger than add, it may"trim" the graph enough that more toggle would increase the fitness, because toggle

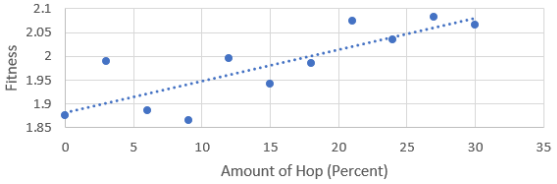
would add the edges in appropriate locations, but not overly connect the graph. This second option is similar to what is seen in the results of the experiments on toggle (Figure 4.1) and the results from the add/delete experiment (Figure 4.7), suggesting that toggle will tend to add more edges between vertices than it removes. Given that toggle adds more edges than it removes, it will tend to create graphs that have a larger size. The null experiment (Figure 4.1(u)) shows that too much toggle starts to under-perform, but with a medium about of toggle, toggle performs very well.



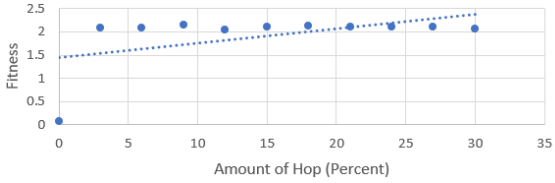
(a) Run 1 Group 1



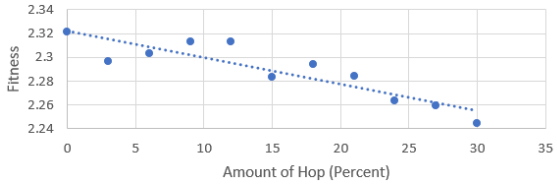
(b) Run 2 Group 1



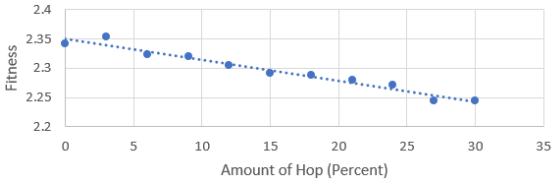
(c) Run 1 Group 2



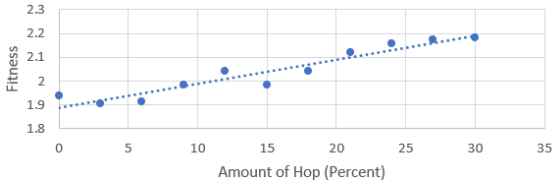
(d) Run 2 Group 2



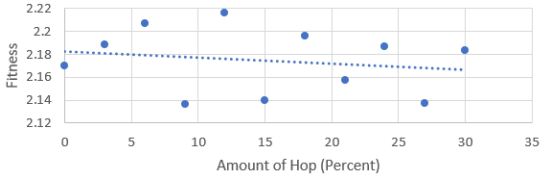
(e) Run 1 Group 3



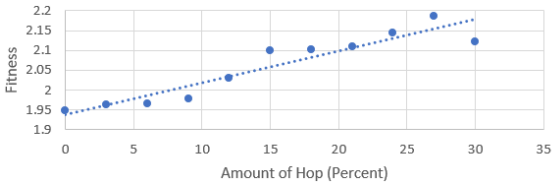
(f) Run 2 Group 3



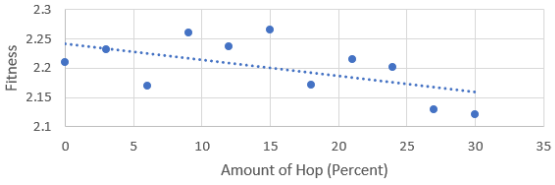
(g) Run 1 Group 4



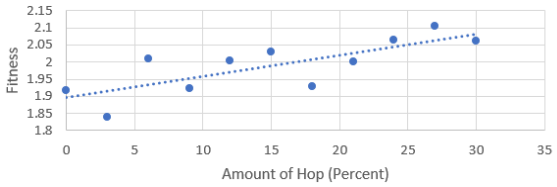
(h) Run 2 Group 4



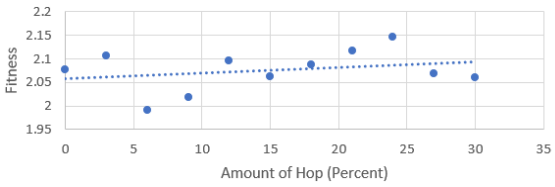
(i) Run 1 Group 5



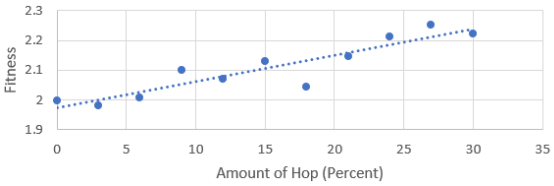
(j) Run 2 Group 5



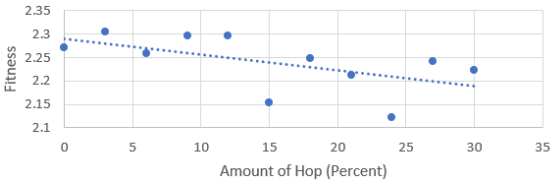
(k) Run 1 Group 6



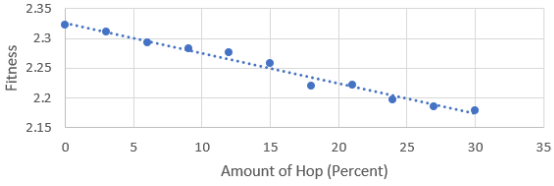
(l) Run 2 Group 6



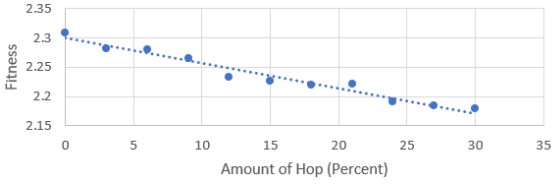
(m) Run 1 Group 7



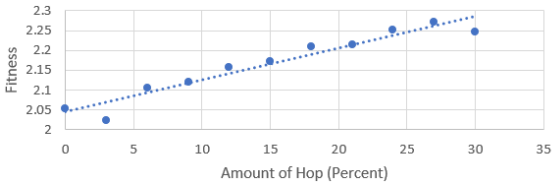
(n) Run 2 Group 7



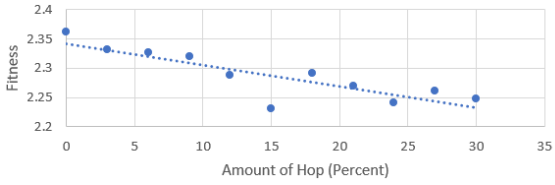
(o) Run 1 Group 8



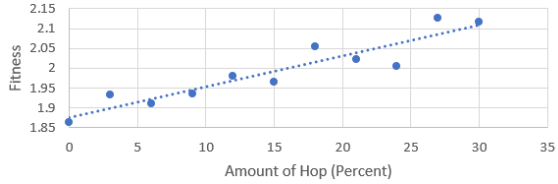
(p) Run 2 Group 8



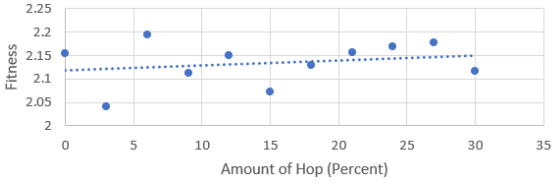
(q) Run 1 Group 9



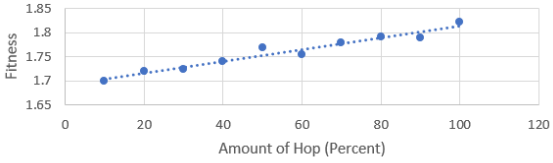
(r) Run 2 Group 9



(s) Run 1 Group 10



(t) Run 2 Group 10

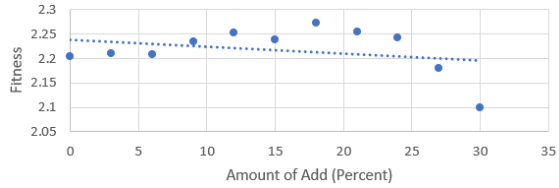


(u) Null Experiment

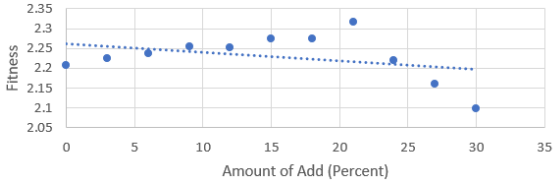
Figure 4.2: Hop Experiments

Hop's experimental results (Figure 4.2) show that hop has a large impact of the fitness. If the fitness is larger, more hop decreases the fitness, while if the fitness is smaller, the

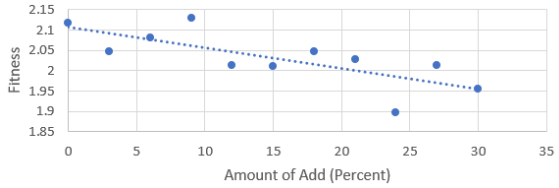
fitness increases. When there is a more of toggle, local toggle and add, and less delete, more hop tends to increase the fitness, to this range, while when there is less toggle, local toggle and add, and more delete, more hop tends to decrease the fitness, bringing it to this range. This suggests that hop creates harder to colour graphs when editing denser graphs (graphs of a larger size, number of edges, when compared its vertices, the number of vertices), while making graphs that are easier to colour when editing sparse graphs (graphs of a smaller size, when compared its density). This could explain why it tends toward a lower fitness, since dense graphs tend to not be as hard to colour. This suggests that hop is very helpful when making complex graphs for dense graphs, but not as useful for sparse graphs. The null experiments (Figure 4.2(u)) show that hop is not able to increase the fitness of the graph very well on its own. Since it improves the fitness with more hop, this suggests that the original graph may be more dense then would be desired for hard to colour graphs. However, since the performance is rather poor, this may not be the case. It may just suggest that the original graph is not very complex at all.



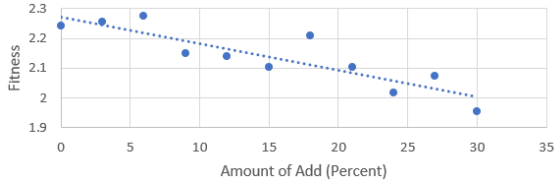
(a) Run 1 Group 1



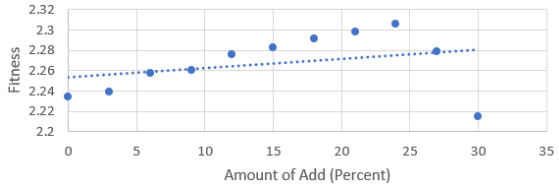
(b) Run 2 Group 1



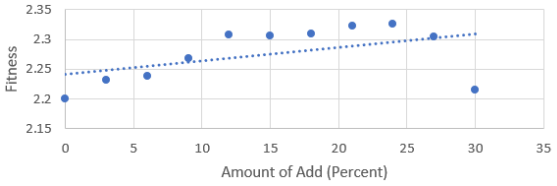
(c) Run 1 Group 2



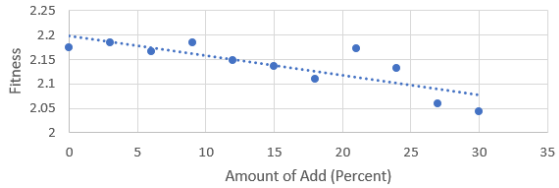
(d) Run 2 Group 2



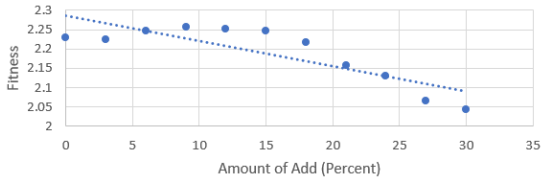
(e) Run 1 Group 3



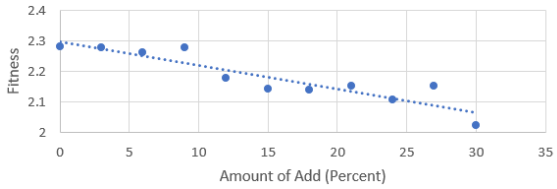
(f) Run 2 Group 3



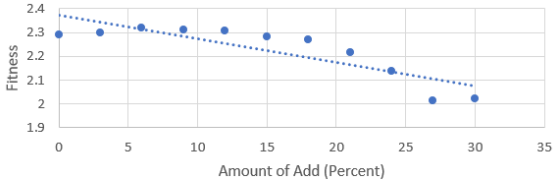
(g) Run 1 Group 4



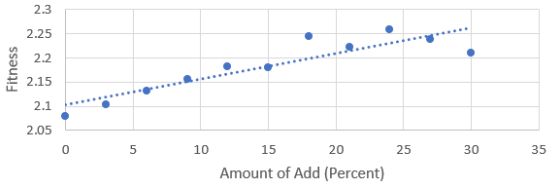
(h) Run 2 Group 4



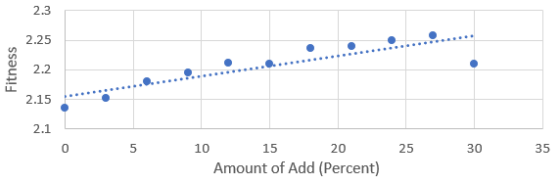
(i) Run 1 Group 5



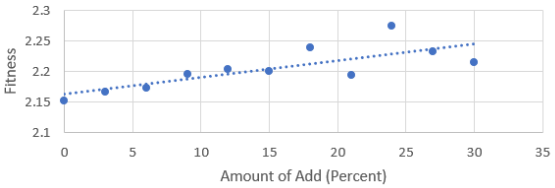
(j) Run 2 Group 5



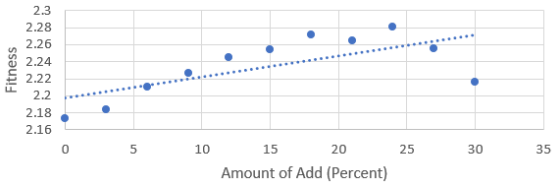
(k) Run 1 Group 6



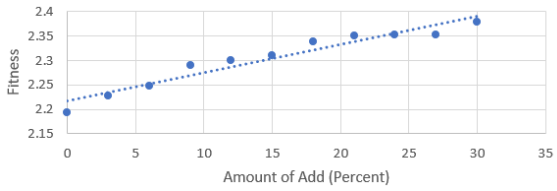
(l) Run 2 Group 6



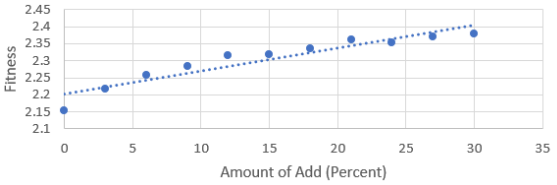
(m) Run 1 Group 7



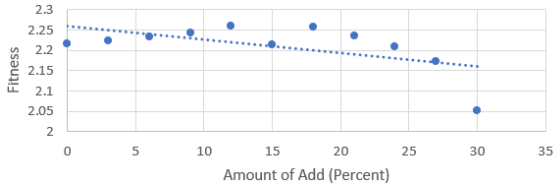
(n) Run 2 Group 7



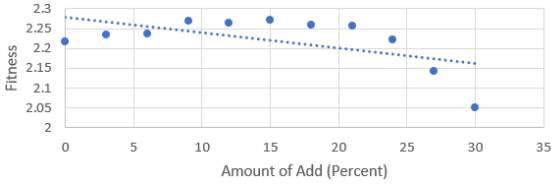
(o) Run 1 Group 8



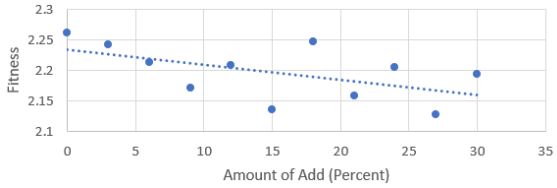
(p) Run 2 Group 8



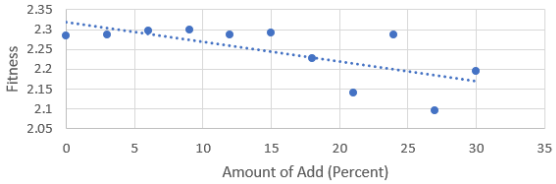
(q) Run 1 Group 9



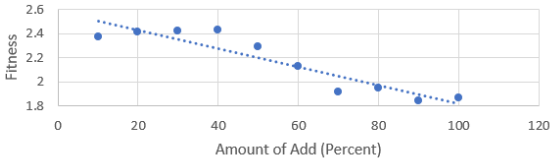
(r) Run 2 Group 9



(s) Run 1 Group 10



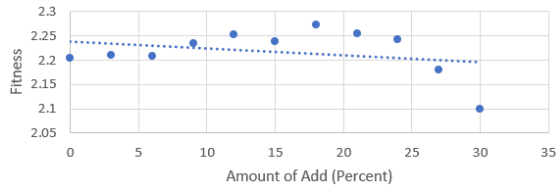
(t) Run 2 Group 10



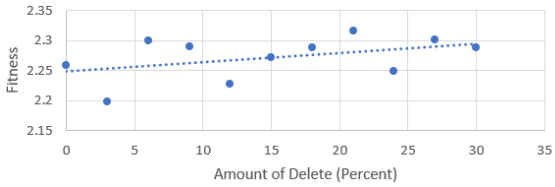
(u) Null Experiment

Figure 4.3: Add Experiments

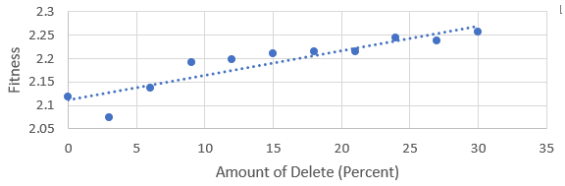
Of all of the results, the results for add and delete edit commands are the least clear – add more so than delete. The results from the experiments testing add are in Figure 4.3. This supports the idea that the more involved the edit commands are, the easier it is to see the characteristics of the edit commands. Add does not appear to have any general performance level that it tends toward. When there is a significant amount of delete, it does perform better, than without. When there is a significant amount of hop, even-though the overall fitness is poor, more add does improve the performance of hop. This continues to support the idea that hop makes harder to colour graphs, when the graphs are denser, while lowers the fitness when the graphs are sparser. When there is a lot of toggle, less add tends to be more effective. This continues to support the idea the toggle tends to create denser graphs. The null experiment (Figure 4.3(u)) shows that a lot of add is not effective, but a small amount of add is effective. This suggests that the graph can be improved by adding a small amount of connections, but too many causes the graph to become too dense. Add seems to have the same amount of impact when the fitness is poor as it does when the fitness is good.



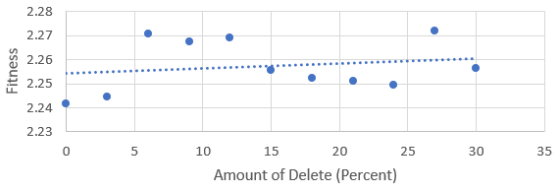
(a) Run 1 Group 1



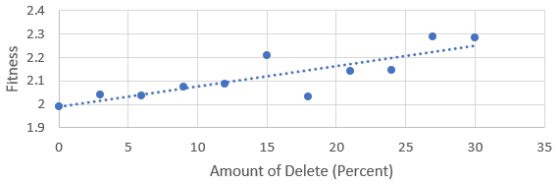
(b) Run 2 Group 1



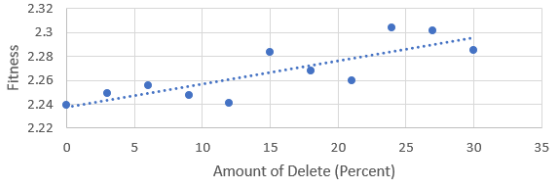
(c) Run 1 Group 2



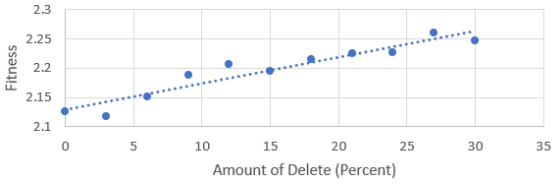
(d) Run 2 Group 2



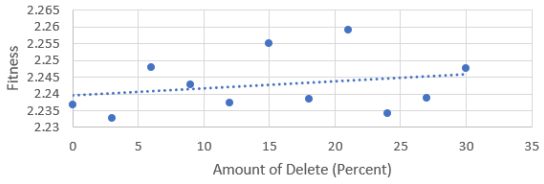
(e) Run 1 Group 3



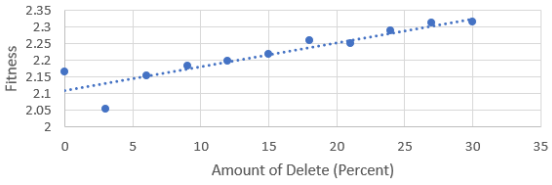
(f) Run 2 Group 3



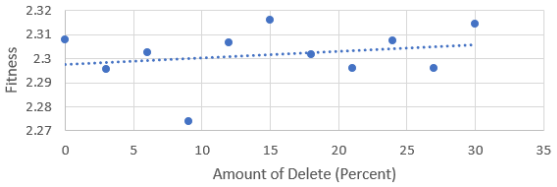
(g) Run 1 Group 4



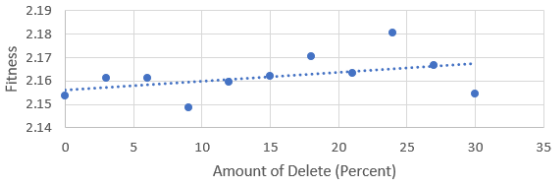
(h) Run 2 Group 4



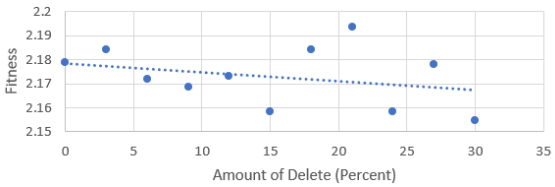
(i) Run 1 Group 5



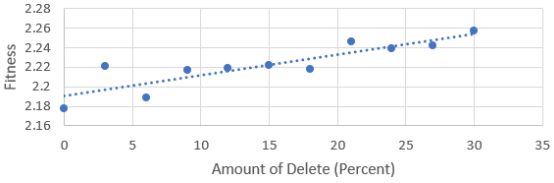
(j) Run 2 Group 5



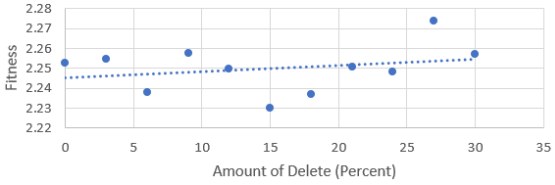
(k) Run 1 Group 6



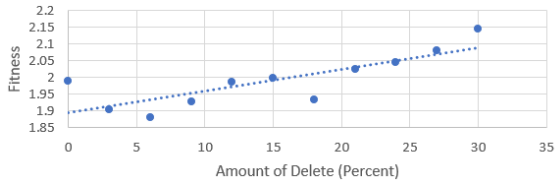
(l) Run 2 Group 6



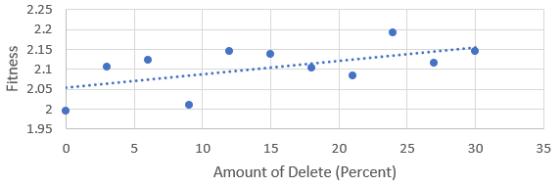
(m) Run 1 Group 7



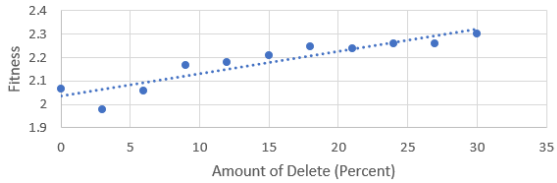
(n) Run 2 Group 7



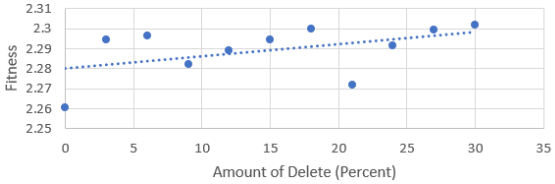
(o) Run 1 Group 8



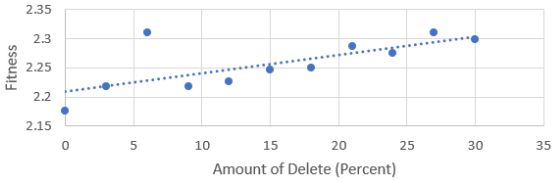
(p) Run 2 Group 8



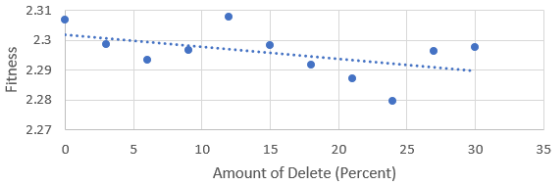
(q) Run 1 Group 9



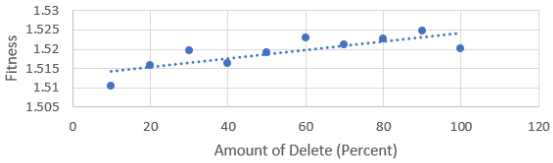
(r) Run 2 Group 9



(s) Run 1 Group 10



(t) Run 2 Group 10

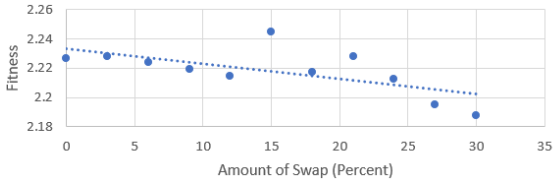


(u) Null Experiment

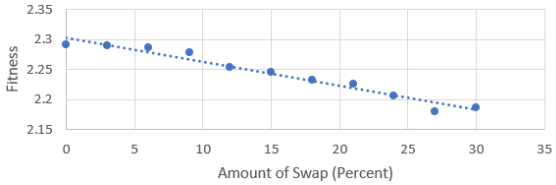
Figure 4.4: Delete Experiments

The results of the experiments on the delete command show that in almost every case, delete increases the performance of the evolutionary algorithm. This suggests that the delete edit command can be used to increase the performance of any chromosomal regulatory sequence. Since the delete operator decreases the size of the graphs that are generated, it suggests that all of the edit commands cause the graphs that are generated to be more dense than is desired for hard to colour graphs. The null experiment (Figure 4.4(u)) shows that delete, on its own, does not increase the performance of the genetic algorithm effectively. This may be because the graphs that are generated must remain connected in-order to be applied. The delete operator may have a lot of times where, if applied, it would separate the graph into two independent graphs, limiting the ability for delete to operate on the graph. The performance of the genetic algorithm does not seem to impact the ability of delete to improve the chromosomal regulatory sequence. Delete always seems to be able to improve

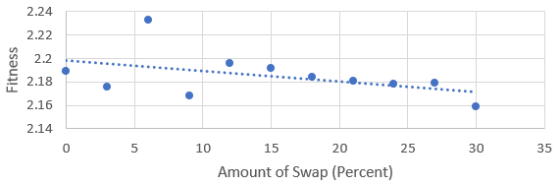
the performance of the genetic algorithm. This suggest that delete should be used later to optimize the parameter settings.



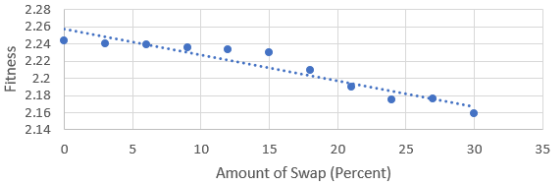
(a) Run 1 Group 1



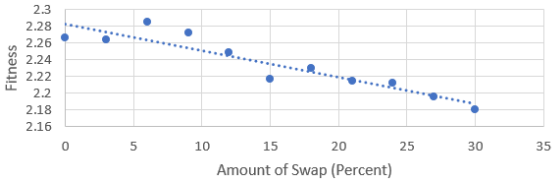
(b) Run 2 Group 1



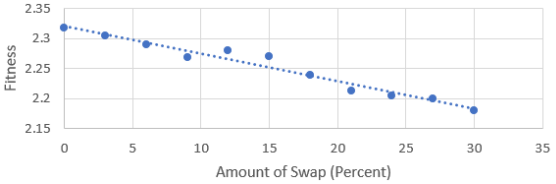
(c) Run 1 Group 2



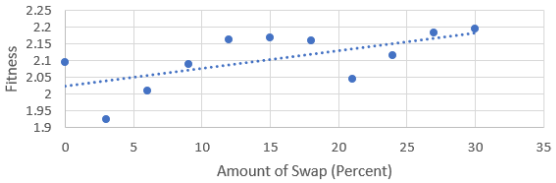
(d) Run 2 Group 2



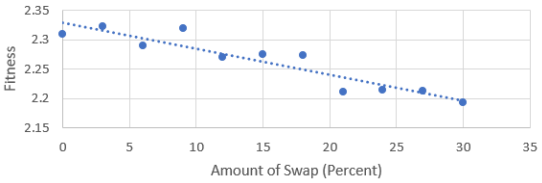
(e) Run 1 Group 3



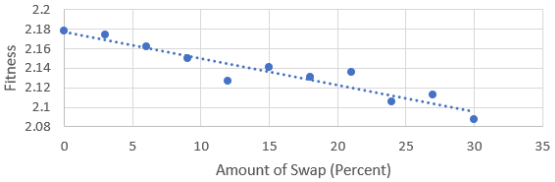
(f) Run 2 Group 3



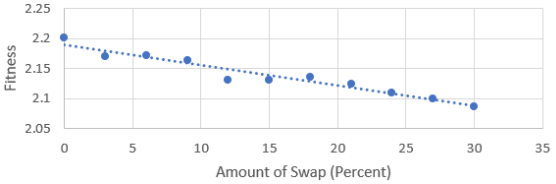
(g) Run 1 Group 4



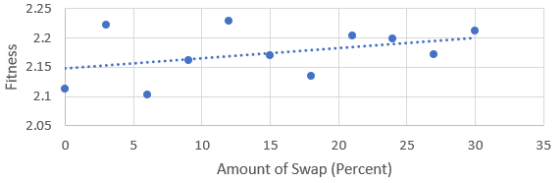
(h) Run 2 Group 4



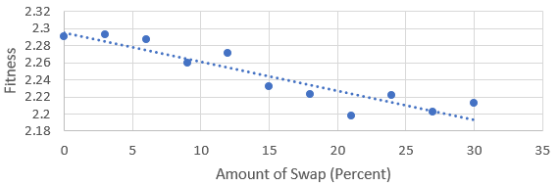
(i) Run 1 Group 5



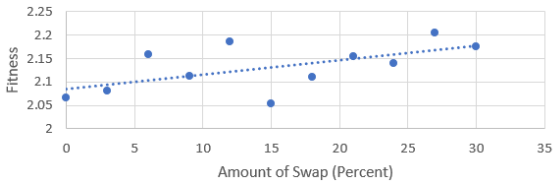
(j) Run 2 Group 5



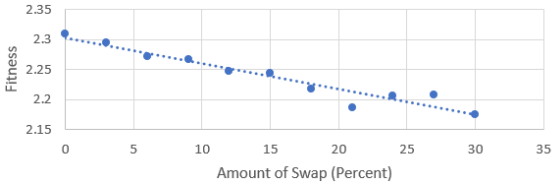
(k) Run 1 Group 6



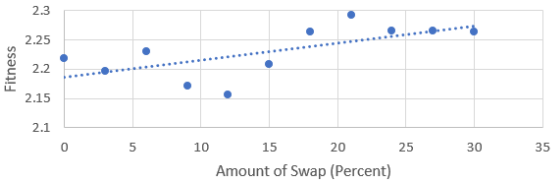
(l) Run 2 Group 6



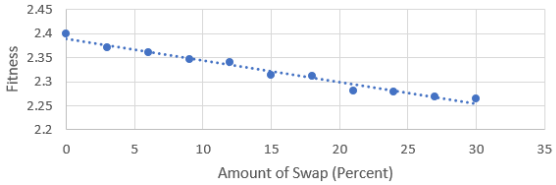
(m) Run 1 Group 7



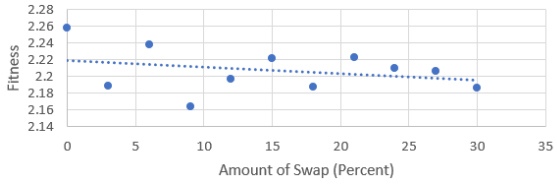
(n) Run 2 Group 7



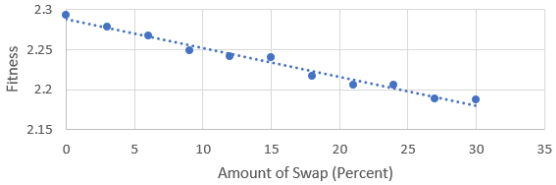
(o) Run 1 Group 8



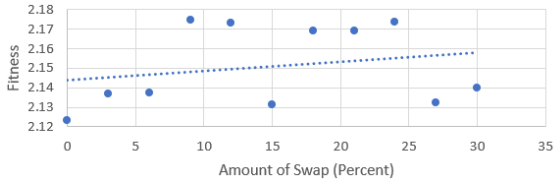
(p) Run 2 Group 8



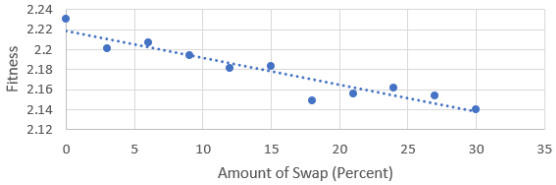
(q) Run 1 Group 9



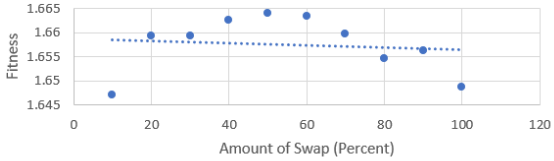
(r) Run 2 Group 9



(s) Run 1 Group 10



(t) Run 2 Group 10

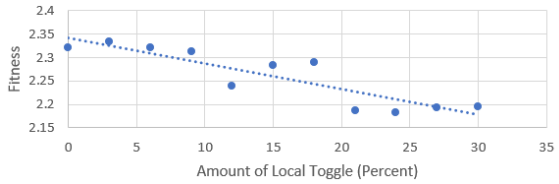


(u) Null Experiment

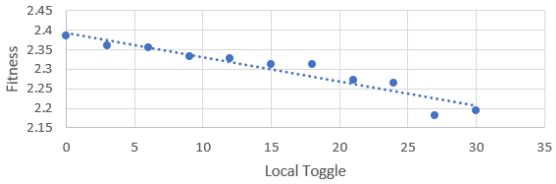
Figure 4.5: Swap Experiments

Swap has interesting results. The first run, shows some decrease and some increase in performance, but the second run shows consistent decrease in performance. The results of the first run of experiments suggests a high co-dependency in the variables used. The results from the second run of experiments suggest that if more swap was used, it would continue to decrease the fitness. There is a little bit of evidence that suggests that if the fitness is lower before swap is added, the effect that adding swap has, is less. One reason for why swap has this consistent behaviour is that swap tends to cause a large amount of change in the

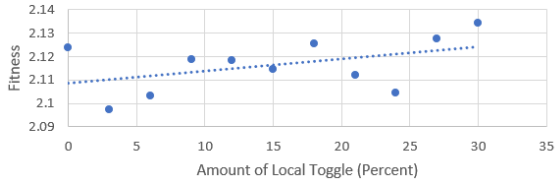
general shape of the graph. Hard to colour graphs tend to have very complex structures, and the large changes made by swap, may brake down these complex structures. This suggests that if you wanted a large interconnected graph with a simple structure, swap may be very useful. The null experiment (Figure 4.5 (u)) shows that swap is not effective at making hard to colour graphs on its own.



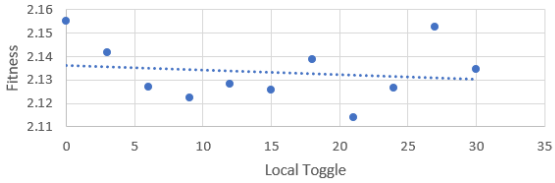
(a) Run 1 Group 1



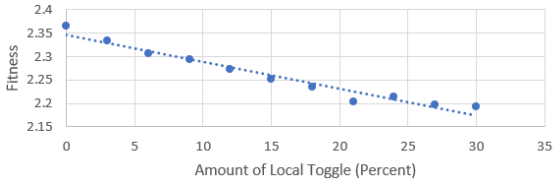
(b) Run 2 Group 1



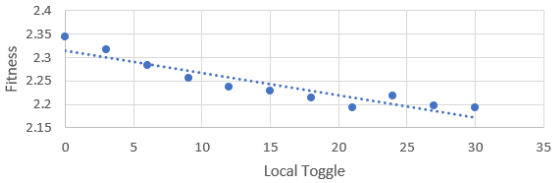
(c) Run 1 Group 2



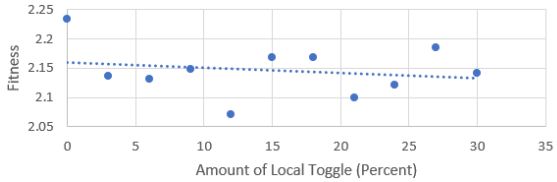
(d) Run 2 Group 2



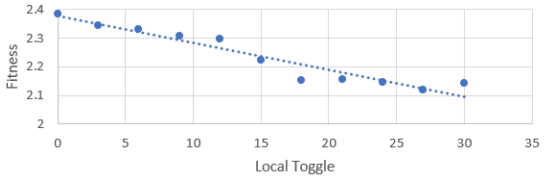
(e) Run 1 Group 3



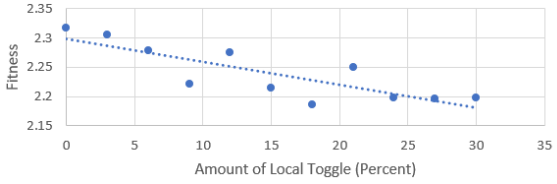
(f) Run 2 Group 3



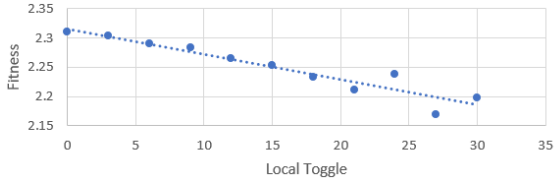
(g) Run 1 Group 4



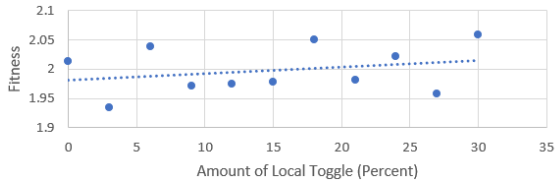
(h) Run 2 Group 4



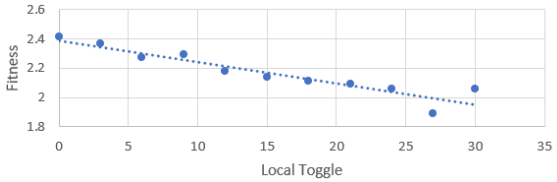
(i) Run 1 Group 5



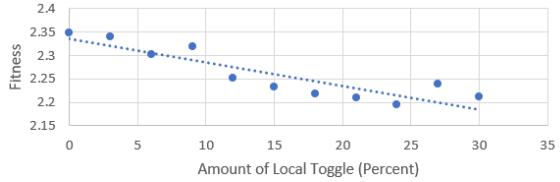
(j) Run 2 Group 5



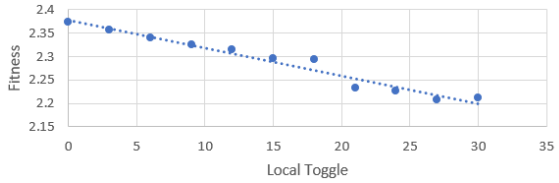
(k) Run 1 Group 6



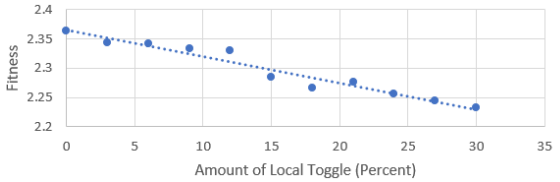
(l) Run 2 Group 6



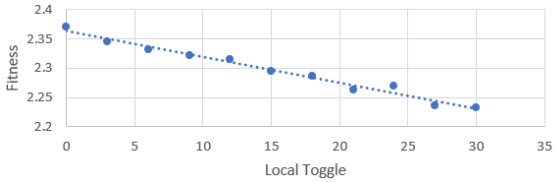
(m) Run 1 Group 7



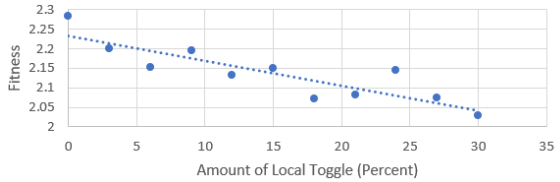
(n) Run 2 Group 7



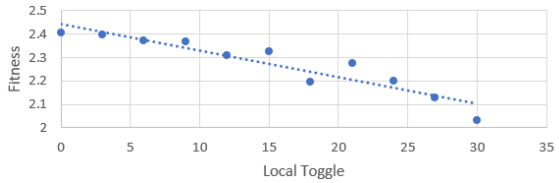
(o) Run 1 Group 8



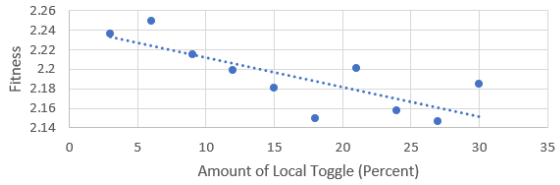
(p) Run 2 Group 8



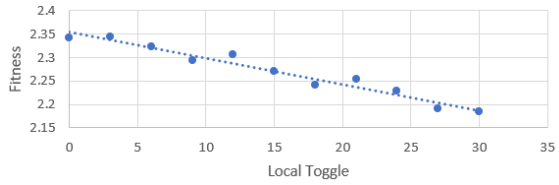
(q) Run 1 Group 9



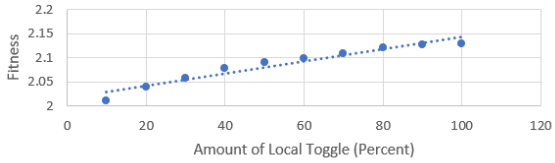
(r) Run 2 Group 9



(s) Run 1 Group 10



(t) Run 2 Group 10

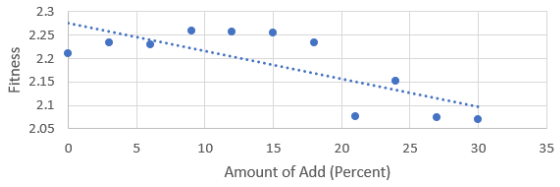


(u) Null Experiment

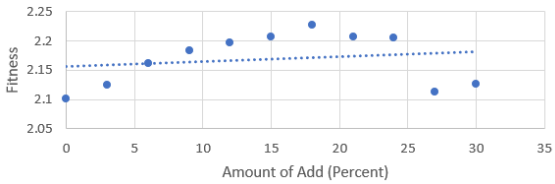
Figure 4.6: Local Toggle Experiments

Local Toggle's results (Figure 4.6) are similar to swap's results in two ways: the second run of experiments are much clearer than the first run of experiments; and in the second

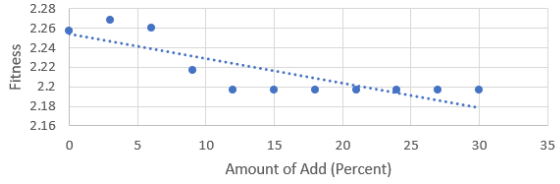
run of experiments an increase in the amount of local toggle results in a decrease in the performance of the genetic algorithm. The second run of experiments shows that the fitness of the graphs tend toward 2.0 to 2.133 and sometimes lower. When there is more delete, local toggle has less of a negative impact on the fitness than when there is less delete. This suggests that like toggle, local toggle tends to add more connections than it removes, but the results suggest it does so to a larger degree than toggle. This makes sense given that graphs that are less dense, tend to be harder to colour. Given a sparse graph, there are a limited number of connected vertices. This means that the chance of two points being connected is unlikely. The chance of three points in that graph forming a cycle would be even more unlikely. This means that when local toggle is applied, it tends to add connections, much more often than it tends to remove them. Adding these edges will form cycles of length three, which are also cliques. This means that each point must be coloured different colours. If done enough times, this will decrease the complexity of the graphs, making it easier to colour for graph colouring algorithms. The null experiment (Figure 4.6(u)) shows that local toggle can perform okay on its own.



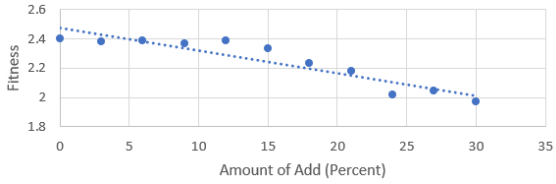
(a) Group 1



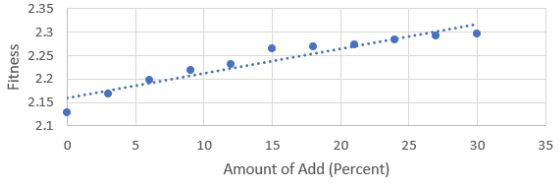
(b) Group 2



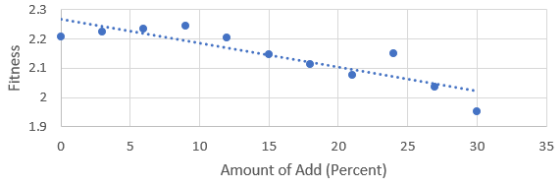
(c) Group 3



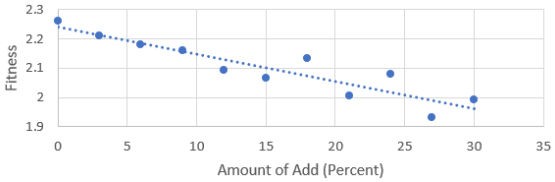
(d) Group 4



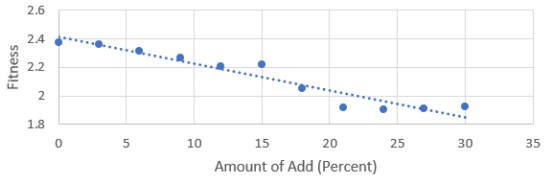
(e) Group 5



(f) Group 6



(g) Group 7



(h) Group 8

Figure 4.7: Add/Delete Experiments

There is high evidence, from the results of the experiments that tested the general characteristics of the edit commands, that the fitness function could be linear, suggesting that LASSO regression may be helpful in optimizing the chromosomal regulatory sequence, but it also shows that the different parameters act in similar ways, causing a high covariance in the results. It appears that a large amount of swap tends toward the lowest performance, then local toggle, followed by hop and toggle. Since add and delete change the graphs even less that toggle, it is assumed that they can create harder graphs than toggle.

Optimal Ratios and Optimal Individuals use this information to order the way in which they optimize the parameters. The following order is used: Swap, Local Toggle, Hop, Toggle, Delete, Add. It orders the edit chromosomes based on the fitness of the graphs that seem to create. Add and Delete are placed after toggle because of the impact they have on changing the size of the graph. The reason Delete is used after Toggle instead of Add, is that with a lot of toggle, delete performs better than add. Toggle is thought to increase the size of the graphs generated. Decreasing the size of the graphs first, before using add gives the add the best chance to have a positive impact on the graph.

There is some evidence that the genetic algorithm could be improved by using different edit commands at different stages in the evolutionary process. This suggests that applying shapes to the genetic algorithm may improve its performance. One possible shape that could be applied to the genetic algorithm is a shape that restricts how the edit chromosomes are applied to the graph when evaluating their individual fitness. This shaping would order the edit chromosomes when applying them to the graph causing the edit chromosome to first use the swap and local toggle operators, then hop, toggle, delete followed by add. The null operator would be applied last. The swap and local toggle edit commands may generate a variety of mildly hard to colour graphs, that then can be improved by hop and toggle. These graphs would then be "trimmed" by delete and add. This may be more effective than applying all of the edit commands at the same time using the original ordering of the edit chromosome. After the fitness of the edit chromosome as been tested, the edit chromosome

would be reordered back to its original order allowing the genetic algorithm to evolve as normal.

The testing of individual edit commands has shown that each of the edit commands seems to have have some general characteristics than could potentially be exploited in the future by applying different shapes to the genetic algorithm. They also show the importance of using appropriate parameter settings.

4.3 The Different Parameter Optimization Techniques

When testing different parameter optimization techniques the computational load of each technique is known, but the performance of each of these techniques is not known; particularly Optimal Ratios and Optimal Individuals, which are new techniques. Genetic Algorithms have very complex solution spaces, meaning that more restrictive searches of the solution spaces may severely under-perform, suggesting that the parameter optimization techniques that have a larger computational load may be more useful, if their results are worth it. If the techniques that are less computationally intense perform well, there may be an appropriate middle ground for computationally concious parameter optimization. The use of the parameter optimization is very important to keep in mind, when determining the appropriate balance between performance and computational load.

Four parameter optimization techniques are tested, two of which are classical and two of which are new. The classic techniques are particle swarm optimization and differential evolution, and the two new techniques, beam optimization using optimal ratios and beam optimization using optimal individuals.

4.3.1 Particle Swarm Optimization

Of all the parameter optimization techniques being used in this study, particle swarm optimization is the oldest. It is the technique with the largest computational load being used. Even though it is not run as long as it would normally be run, it is able to produce effective results, given the proper boundary conditions. The three experiments that were run had three different boundary conditions, each having interesting results.

When there is no restriction on the boundary of the particle swarm optimization, the particles move around sporadically, usually causing one edit command to be isolated with the null operator. Just like in the results of the null experiments, the best performing chromo-

somal regulatory sequences isolate toggle with null. Because of the sporadic behaviour, this rarely occurs, requiring a large number of iterations to find effective combinations of toggle. This method is unable to fine tune more parameters than the one that is most effective by itself, limiting its performance. The best performing chromosomal regulatory sequence that it found, with a fitness of 2.437 is 34.71%, toggle and 65.30% Null, with zero likelihood of any of the other edit commands.

The first boundary condition for particle swarm optimization does not work effectively. This boundary condition limits the range of the particles, but does not stop their momentum. Out of the three different experiments, this particle swarm optimization performs the worst. The best chromosomal regulatory sequence it finds is 19.43% toggle, 0.0012% hop, 18.44% add, 22.93% delete, 1.012% swap, 0% local toggle and 38.20% null, which has a fitness of 2.426. This version quickly finds rather effective chromosomal regulatory sequences, but gets stuck very quickly. This limits its ability to improve beyond its initial movements. One possible fix for this is to project the movement onto the boundary. This would require much more sophisticated programming, but may improve the result of the particle swarm optimization dramatically, by allowing the particles to move along the edges of the solution space. Because there is a lack of movement in the particles, the program can also be effectively increased by removing double runs of chromosomes with the same parameter settings, this would decrease the run time, allowing the particles to move more freely.

The third set of boundary conditions is the most effective. This boundary condition causes particles to"stick" to the boundaries, losing any momentum it has. Within 10 iterations, which still takes over 42 hours, it EC-converges to the most effective chromosomal regulatory sequence out of all of the particle swarm optimizations, with a fitness of 2.443. This chromosomal regulatory sequence is 15.11% toggle, 1.02% hop, 34.28% add, 28.39% delete, 0.76% swap, 0% local toggle and 20.44 percent null. This is much quicker than expected, since normal particle swarm optimizations take thousands of iterations to EC-converge effectively. One reason why the particle swarm optimization may EC-converge so quickly is that the genetic algorithm may have a large subsection of the solution space is similarly optimal. Since we are just trying to find a rather effective parameter setting, this is desirable.

Particle Swarm optimization works much more effectively than expected, EC-converging to effective solutions rapidly. Even with these results, it still takes almost 2 days to test any given fitness function. The optimal likelihood of each of the edit commands make sense, for the fitness function used, given the research on each of the individual edit commands.

The particle swarm optimization does not work when left alone, but when given appropriate boundary conditions, it performs much better than expected.

4.3.2 Differential Evolution

Differential Evolution is slightly newer that particle swarm optimization. It has a tendency to EC-converge more quickly than particle swarm optimization. With all three boundary conditions, the agents (what particles are called in differential evolution) EC-converge to solutions much more quickly. The boundary conditions do not seem to make as much of an impact in differential evolution than they do in particle swarm optimization.

The differential evolution with no boundary condition takes a long time to EC-converge, continually increasing the fitness of the genetic algorithm. It continues to EC-converge to more effective parameter settings than particle swarm optimization. In 10 days it is able to EC-converges to a fitness of 2.446 which is better than any result that the particle swarm optimization is able to achieve. In 10 iterations it is able to achieve a fitness of 2.424, with a chromosomal regulatory sequence of 43.24% likelihood of add, 53.05% likelihood of delete, and a 3.71% likelihood of null the other edit commands have a likelihood of zero. This is a pretty effective solution, however, it EC-converges much slower than the particle swarm optimization with the second boundary conditions. By the end of the 40 iterations is does not appear to have finished EC-converging to the optimal chromosomal regulatory sequence.

The first set of boundary conditions, while forcing the agents to stay within the boundary of the probability simplex, is only able to achieve a fitness of 2.345 after 10 iterations and after 40 iterations, it is only able to attain a fitness of 2.375. It achieved this with a chromosomal regulatory sequence of 17.28% toggle, 12.66% hop, 15.44% add, 23.80% delete, 4.27% swap, 1.65% local toggle and 24.90% null. This boundary condition seriously restricts the performance of differential evolution. This may be because it highly encourages the algorithm to explore the inside of the probability simplex, instead of around the boundary, forcing edit commands like swap and local toggle to be overused. This boundary condition seems to limit the ability of differential evolution to EC-converge to optimal solutions.

The second boundary condition, which has even more restrictions, performs better than the first set of boundary conditions, but is not as effective as differential evolution that does not have any boundary conditions. In 10 iterations it has a fitness of 2.415, but is only able to increase to best agent's fitness to 2.429 by the 40th iteration. Just like the first set of boundary conditions, the second type of boundary conditions cause differential evolution to

EC-converge to quickly, limiting its ability properly search the solution space.

Differential evolution appears to be pretty effective when left unmodified. It does not EC-converge as quickly to optimal solutions as particle swarm optimization, but shows signs of stopping. Given enough time, differential evolution will probably out-perform the particle swarm optimization, but this requires much more computing power and time. The boundary conditions do nothing but hinder differential evolution. Differential evolution does not need the boundary conditions to keep the agents near enough to the boundary to be effective. These boundary conditions actually cause the differential evolution to under-perform and EC-converge too quickly.

4.3.3 Beam Optimization

The results for beam optimization are rather unsatisfactory. Both using optimal ratios and optimal individuals results in finding a poor chromosomal regulatory sequence. In both cases the under-performing edit commands are over-represented, while the better edit commands are under-represented.

Optimal Ratios

When optimizing using optimal rations, toggle is under-represented, when compared to local toggle and hop. When toggle is factored in swap was at zero percent, local toggle was at 94.05% and hop was at 5.95%. When introducing toggle all of the chromosomal regulatory sequences are mostly made up of toggle and local toggle. This creates graphs that tend to be denser, since toggle and local toggle tend to add more edges than they remove. This makes hop perform better and encourages hop to be a larger part of the chromosomal regulatory sequence. This may be why toggle is under-represented. Even with these challenges, beam optimization using optimal ratios is able to find a chromosomal regulatory sequence that has a fitness of 2.408, which is pretty good. However it took longer to achieve this than it took particle swarm optimization or differential evolution to find a fitter chromosomal regulatory sequence.

Optimal Individuals

Optimal Individuals really under-performs. This is mainly because the optimal amount of swap is 49%. Given this high optimal amount of swap the optimal amount of local toggle is 49.98%, leaving just 1.02% of the chromosomal regulatory sequence left for the other

operators. Based on the results from the individual experiments on the edit commands, this severely over-represents swap and local toggle and under-represents delete, add, and toggle. The eventual best fitness is about 2.05.

It does not seem that beam optimization is effective for this parameter optimization task. In order to set up the process, a lot of background research needs to be done on the edit commands that are being used and it does not appear to have a smaller computational load or better results than differential evolution or particle swarm optimization, which do not need the extensive setup.

4.4 Conclusion

The results of the individual experiments on the edit commands show that the different edit commands have very different impacts on how they shape the graphs. It showed that some edit commands perform better on denser graphs, while others performed better on sparser graphs. It seems to suggest there may be a linear relationship between the variable and that LASSO Regression could be used to optimize the parameters of the chromosomal regulatory sequence. The parameter optimization results show that when applied to probability simplexes, particle swarm optimization benefits from boundary conditions, while differential evolution is hindered by boundary conditions. The parameter settings are shown to be very important. There is a large amount of variation between different chromosomal regulatory sequences. One of the worst chromosomal regulatory sequence had a performance of 2.020, with a standard deviation of 0.2978, while one of the best chromosomal regulatory sequences had a performance of 2.446.

The results from edit command experiments suggest that some of the edit commands are more effective earlier on in the process, while others are better later in the process. This brought about the idea for beam optimization, which turned out to be very ineffective; however, using shaping within the genetic algorithm may make the genetic algorithm more effective at finding hard to colour graphs for a given graph colouring algorithm.

Chapter 5

Conclusion And Further Work

As computers become more advanced, there are more applications of graps. Each of these applications desire different types of graphs, with different characteristics. As the graphs become larger, creating these graphs becomes harder. The genetic algorithm used in this thesis has been used in the past to create graphs with specific structures. This thesis mainly looks at how the parameters of the genetic algorithm can be setup effectively. Part of this process analyses the effect of the number of iterations, the population size, and the mutation number. Part of this process looks at how the edit commands of the genetic algorithm affect the structure of the graphs they created. Part of it looks at how the chromosomal regulatory sequence can be optimized.

As this genetic algorithm continues to be used for different applications, the characteristics of each edit command could be taken into account when creating graphs. As algorithms are developed to find optimal solutions for graphs, hopefully genetic algorithms can start being used to create graphs that can be used to find the weaknesses in the algorithms, allowing them to be improved.

5.1 The Number of Iteration, Population Size and the Number of Mutations

The genetic algorithm, given the fitness function and graph colouring algorithm used in this thesis performs better with a larger population size and mutation number. This may be different given a different fitness function or graph colouring algorithm. One could test different graph colouring algorithms, using the same fitness function, and determine if a large

population and mutation number is still advantageous.

The number of iterations adjusts how computationally intense the genetic algorithm is. Once the chromosomal regulatory sequence has been set, the genetic algorithm could be run again with a larger number of iterations, to see if it can create even harder to colour graphs, given enough time.

5.2 General Characteristics of the Edit Commands

Some interesting characteristics are discovered for some of the edit commands. These characteristics have some specific implications on how the edit commands manipulate the graphs. As this genetic algorithm is being used to create graphs that have specific structures, some of the results that were discovered about the operators may be used to better create said graphs.

In 1999, Eiben suggested that different edit commands are more effective at different stages of the evolutionary algorithm (Eiben, 1999). Some of the edit commands, like swap, seem to be more effective at making large changes to graphs, but are not as effective at "fine tuning" the graph; while other edit commands, like add and delete, are more effective at "fine tuning" the graph, but not as effective at making large changes to the structure. Currently the genetic algorithm applies the edit commands with the same likelihood throughout the entire process of the genetic algorithm. Ashlock and Ashlock (2014) have shown that adding a shape to evolutionary algorithms can improve their performance.

DEFINITION A *Shape* is a restriction on the way a representation in an evolutionary computation system may be instantiated.

The results of this thesis suggest that a new type shape, called a *probabilistic shape* may be effective if applied.

DEFINITION A *probabilistic shape* is a restriction on the probability of things occurring within an evolving structure.

Future work could be done to see if the algorithm could be improved by applying a probabilistic shape to give the edit commands that make large changes a higher probability in the beginning of the process, while later in the process increasing the probability of the edit commands which are better at "fine tuning" the graphs.

There are many shapes one could apply to the genetic algorithm that impact the order in which the edit commands are applied. One possible probabilistic shape could force the chromosomal regulatory sequence to modify as population evolves. The chromosomal reg-

ulatory sequence would begin with a larger likelihood of edit commands which make large changes. As the genetic algorithm evolves, the chromosomal regulatory sequence slowly changes, favouring the edit commands that are more effective at small changes. This would require some transition mechanism that may increase the computational load of optimizing, but may significantly increase the performance once optimized.

Another shape which could affect the order in which edits are made to a graph is to have a number of multiple different edit chromosomes that are individually applied in a specific order, each of which are governed by there own chromosomal regulatory sequence. This would allow larger changes to be made early in graph editing and smaller changes as the graphs become more developed. This shaping would require multiple chromosomal regulatory sequences to be optimized, increasing the computational load of parameter optimization, but may increase the performance of the genetic algorithm.

Another shape that could be applied, which is discussed in the results section, affects the edit chromosomes that are created by the chromosomal regulatory sequence. The creation and evolution process of the genetic algorithm would be the same, the only difference is that when the edit chromosome is used, it would be ordered before applied. One possible way the edit chromosomes could be shaped is to order the edits from edits that are better in the early stages of modifying the graph to edits that are better in the later stages. For this particular case the edit swap would happen first, then local toggle, then hop, followed by toggle, delete, add and finally null. This shape would not increase the computational load of parameter optimization, in fact, the same parameter optimization techniques used in this thesis could be used on the shaped genetic algorithm.

Shaping the genetic algorithm may improve the performance specifically for graph colouring, but may not be effective for different applications of this genetic algorithm. Ashlock and Ashlock (2014) have shown that with enough knowledge about the genetic algorithm, shapes can be applied appropriately and improve the performance, however they must be applied appropriately, requiring more preliminary knowledge on the problem before shaping the algorithm. These different shapes may improve the genetic algorithm, but some of them may increase the computational load of the parameter optimization too much. Discerning what is the appropriate trade-off between computational load and performance of the genetic algorithm, when shapes are being applied could be looked at in the future.

5.3 Optimizing the Chromosomal Regulatory Sequence

Michalewicz *et al.* (1996) claim that proper setup is required in order for genetic algorithms to be used effectively. The variety of results that are found when trying to optimize the parameters of this genetic algorithm support that claim. The new techniques that were tried did not work effectively, supporting Eiben's *et al.* (1997) idea that systematically searching the possible parameter settings is not effective. As genetic algorithms become more computationally complex, discerning appropriate ways in which to optimize their parameters becomes important as well.

The results from the particle swarm optimization and differential evolution tests show that they may be able to help optimize genetic algorithms that are very computationally intense, as long as they have appropriate constraints. Using a particle swarm optimization that had the second boundary conditions and ten iterations was the most effective way of optimizing the parameters of the fitness function. While introducing boundary conditions to differential evolution negatively impacted its performance. For solution spaces that lie on probability simplexes, having a boundary condition for a particle swarm optimization that projects the momentum onto the boundary may be effective. As the uses of particle swarm optimization and differential evolution continue to be explored, different boundary conditions may be needed for different situation spaces. The uses of particle swarm optimization and differential evolution on these different solution spaces could be explored.

5.4 Testing Algorithmic Graph Solvers

The majority of the research in this thesis was based around the question of whether or not the genetic algorithm could be used to find hard to colour graphs for graph colouring algorithms. Some of the best parameter settings were able to find graphs with a fitness of about 2.45, while the worst parameter settings had a fitness of 1.4. A fitness of 1.4 is okay, but a fitness of 2.45 is really good. This means the average number of colours used by the greedy colouring algorithm was 2.45 times more then the lowest number of colours used by the graph colouring algorithm. The genetic algorithm is pretty effective at finding hard to colour graphs for the greedy colouring algorithm. To further test the graph colouring algorithm, the genetic algorithm could be run with a larger number of iterations using the optimal chromosomal regulatory sequence. Testing the graph colouring algorithm with the optimal chromosome with a larger number of iterations may improve the genetic algorithm,

resulting in even harder to colour graphs.

This technique could also be used to find harder to colour graphs for other classical graph colouring algorithms, like the CS method or the LF method; as well as testing modern graph colouring algorithms, like Golberg *et al.*'s method (1988) or Schneider and Wattenhofer's method (2010). The hard to colour graphs for the modern graph colouring algorithm that are discovered by the genetic algorithm may have consistent characteristics. Since modern graph colouring algorithms use characteristics of graphs to help colour them, more algorithms could be discovered that are able to utilize these characteristics to more accurately colour the graph. As graph colouring algorithms continue to be developed this may be an effective tool used to test them. This genetic algorithm may be able to develop not only graph colouring algorithms, but other graph solvers as well.

The effectiveness of the genetic algorithm relies heavily on the fitness function that is used. Coming up with these fitness functions can be difficult, and testing them can be even more difficult. The ability to optimize the parameters quickly to get the best results for a given fitness functions allows fitness functions to be tested quicker, hopefully allowing better fitness functions discovered for specific purposes.

New ways in which graphs can be used are constantly being developed. As these applications continue to be developed, more graphs need to be developed to meet these needs. This genetic algorithm may be able to be adapted to help generate these graphs. Particle swarm optimization with the second type of boundary conditions may reduce the time it takes to find these appropriate graphs.

Alhomidi, M. and Reed, M. (2013) Finding the minimum cut set in attack graphs using genetic algorithms. In *Computer Applications Technology (ICCAT), 2013 International Conference*, 1-6.

Ashlock, D. (2006) Evolutionary computation for optimization and modelling. Springer, New York.

Ashlock, D. (2016) The seven bridges problem [blog post]. Retrieved from https://occupymath.wordpress.com/2016/02/19/can-you-draw-this/

Ashlock, D. (2018) A course in evolutionary computation. *Unpublished lecture notes.*

Ashlock, D. and Barlow, L.A. (2015) A class of representations for evolving graphs. In *2015 Congress on Evolutionary computation (CEC)*, Sendai Japan, IEEE Press, 1295-1302

Ashlock, D., Schonfeld, J., Barlow, L. and Lee., C. (2011) Comparison of Evolved Epidemic Networks with Diffusion Characters. *Proccedings of the IEEE Symposium on the Foundations of Computational Intelligence*, 781-788.

Ashlock, D., Schonfeld, J., Barlow, L. and Lee., C. (2014) Test Problems and Representaions for Graph Evolution. *Processings of the IEEE Symposium on the Foundations of Computational Intelligence*, 38-45.

Ashlock, D. and Shiller, E. (2011) Fitting contact networks to epidemic behavior with an evolutionary algorithm. In: *Proceedings of the 2011 IEEE Symposium on Computational Intelligence in Bioinformatics and Computational Biology*, Piscataway NJ, IEEE Press, 1-8.

Ashlock, D. and Timmins, T. (2016) Adding local edge mobility to graph evolution. *Proceedings of the IEEE 2016 Congress on Evolutionary Computation*, 1594-1601.

Ashlock, W. and Ashlock, D. (2014) Shaped prisoner's dilemma automata. *Proceedings of the IEEE Conference on Computational Inteligence in Games*, 76-83.

Bäck, T., Fogel, D. and Michalewicz, Z. (1997) *Handbook of Evolutionary Computation.* New York: Institute of Physics Publishing Ltd., Bristol and Oxford University Press.

Ballseter, P.J. and Carter, J.N. (2003) Real-parameter genetic algorithms for finding multiple optimal solutions in multi-modal optimization. *GECCO 2003*, 706-717.

Barenboim, L. and Elkin, M. (2009) Distributed $(\Delta+1)$-coloring in linear (in Δ) time. *ACM* (STOC'09), 111-120.

Bonyadi, M.R. and Michalewicz, Z. (2017) Particle swarm optimixation for single objective continuous space problems: a review. *Evolutionary Computation*, 25(1):1-54.

Brélaz, D., (1979) On colouring nodes of a network. *Proc. Cambridge Philos. Soc.*, 37:194-197.

Brooks, R.L. (1941) On colouring the nodes of a network. *Proc. Cambridge Philos. Soc.*, 37:194-197.

Campbell, J.K. (2005) Enumeration and symmetry of edit metric spaces. (Doctoral Dissertation) *Iowa State University.*

Cutello, V., Nicosia, G., and Pavone, M. (2003) A hybrid immune algorithm with information gain for the graph coloring problem. In *Proceedings of GECCO* Berlin, Springer, 171-181.

De Bonet J.S., Isbell Jr., C.L., and Biola, P. (1997) Finding optima by estimating probability densities. *Adv Neural Inf Process Syst*, 10:424-430.

De Jong, K. (1975) *The analysis of the behavior of a class of genetic adaptive systems* (Doctoral dissertation) Dept. Computer Science, Univ. of Michigan, Ann Arbor.

Eiben A.E., Hinterding, R. and Michalewicz Z. (1999) Parameter Control in Evolutionary Algorithms. *IEEE Trans. Evol. Comput.*, 3(2):124-141.

Euler, L. (1735) Šolutio prolematix ad gemetriam situs pertinentis. *Comment. Acad. Sci. U. petrop*, 8:128-140.

Fleurent, C., and Ferland, J.A (1996) Genetic and hybrid algorithms for graph coloring. *Ann Oper Res*, 63:437-461

Fraser, A.S. (1958) Monte Carlo analyses of genetic models. *Nature* 181(4603)208-209.

Reitage, M. and Al-Onaizan, Y. (2017) Beam search strategies for neural machine translation. *Proceedings of the Girst Workshop on Neural Machhine Translation*, 56-60.

Galan, S.F. and Mengshoel, O.J. (2010) Generalized crowding for genetic algorithms. *GECCO'10*

Galinier, P. and Hao, J.K. (1999) Hybrid evolutionary algorithms for graph coloring. *Journal of combinatorial optimization*, 33(9):2547-2562.

Garcia-Piquer, A., Fornells, A., Bacardit, J., Orriols-Puig, A. and Golobardes, E. (2014) Large-scale experimental evalution of cluster representations for multiobjective evolutionary clustering. *IEEE Trans. Evol. Comput.*, 18(1):36-53.

Garey, M.R. and Johnson, D.S.(1976) The complexity of near-optimal graph coloring. *J. ACM*, 23:43-29.

Geller, D.P. (1970) Problem 5713. *Amer, Math. Monthly*, 77:85.

Goldberg, A.V., Plotkin, S.A, and Shannon, G.E. (1988) Parallel symmetry-breaking in sparse graphs. *SIAM J. Dis. Math.*, 1(4):434-446.

Goldberg, D.E (1989) *Genetic algorithms is search, optimization and machine learning.* Reading, MA: Addison-Wesley

Grefenstette, J.J (1986) Optimization of control parameters for genetic algorithms. *IEEE Trans. Systems, Man, Cybern.,* 16(1):122-128.

Hanchate, D.B. and Bichkar, R.S. (2014) Software project scheduling management by particle swarm optimization. *Economics of Knowledge*, 6(4):24-54.

Holland, J.H. (1975) *Adaptation in natural and artificial systems.* University of Michigan Press.

Johnson, D.S. (1974) Approximation algorithms for combinatorial problems. *J.Comp. Syst. Sci.*, 9:256-278.

Jones, B. (2010) Royalty free, printable, blank, united kingdom map with administrative district borders. [Online Image] Bruce Jones Design Inc.

Kennedy J. (1997) The particle swarm: social adaptation of knowledge. *Proceedings of IEEE International Conference on Evolutionary Computation*, 303-308.

Kennedy, J. and Eberhart, R. (1995) Particle swarm optimization. *Proceedings of IEEE International Conference on Neural Networks*, 1942-1948.

Kéry, G. (1964) ON a theorem of Ramsey (in Hungarian). *Matematikai Lapok* 15,204-224.

Kosowski, A. and Manuszewski, K. (2004) Classical coloring of graphs. In M. Kubale *Graph Colorings* Providence, Rhode Island: American Mathematical Society, 1-19.

Koza, J.R. (1998) Genetic programming. *Encyclopedia of Computer Science and Technology.*, 93(24):29-43.

Kubale, M. (2004), Graph colorings. *American Mathematical Society*, ISBE 0-8218-3458-4lan

Kuhn, F. (2009) Weak graph colorings: distributed algorithms and applications. *ACM* (SPAA'09), 138-144.

Kumar, A., Kumar, A., Choudhary, S. and Varde, P.V. (2010) Optimization of binary decision diagram using genetic algorithms. In *Reliability, Safety and Hazard (ICRESH), 2010 2nd International conference*, 168-175.

Lichtblau D(2011) Differential evolution in discrete optimization. *International Journal of Swarm Intelligence and Evolutionary Computation* 1, 1-10.

Matula, B.W., Marble, G. Isaacson, D. (1972) Graph colouring algorithms. In *Graph Theory and Computing*, New York: Academic Press, 109-122.

McEachern, A. and Ashlock, D. (2014) Shape control of side effect machines for DNA classification. *Proceedings of the 2013 IEEE Conference on Compuational Intelligence in Bioinformatics and Computational Biology*, 1-8.

Michalewicz, Z. and Schoenauer, M. (1996) Evolutionary algorithms for constrained parameter optimization problems. *Evol. Comput.*, 4:1-32.

Miller, B. and Goldberg, D. (1995) Genetic algorithms, tournament selection, and the effects of noise. *Complex Systems*, 9:193-212.

Montalvo, I., Izquierdo, J., Pérez-Garcia R., and Herrera, M. (2010) Improved performance of PSO with self-adaptive parameters for computing the optimal design of water supply systems. *Eng Appl Artif Intell*, 23(5):727-735.

Mycielski, J. (1955) On the coloring of graphs. *Coll. Math.*, 3:161-162.

Nikolopoulos, S.D., Papadopoulos, C. (2000) On the performance of the first-fit coloring algorithm on permutation graphs. *Information Processing Letters*, 75:265-273.

Phillips, D.L. (1962) A technique for the numerical solution of certain integral equations of the first kind. *Jounal of the ACM*, 9:84.

Pizzuti, C. (2012) A multiobjective genetic algorithm to find communities in complex networks. *IEEE Transactions on Evolutionary Computation*, 16(3):418-430.

Poli, R. (2008) An analysis of publications of particle swarm optimization applications. *Journal of Artificial Evolution and Applications*, 1-10.

Rechenberg, I. (1973) *Evolutionsstratefie-optimierung technizcher systeme nach prinzipien der biologischen evolution* (Doctoral dissertation) (in German). Fromman-Holzboog.

Reddy, R.D. (1977) *Speech understanding systems: a summary of results of the five-year research effort.* Department of Computer Science.

Robertson, N., Sanders, D., Seymour, P., and Thomor, R. (1997) The four color theorem. *J. Comb. Th. B.*, 70:2-44.

Schneider, J. and Wattenhofer, R. (2008) A log-star distributed maximal independent set algorithm for growth-bounded graphs. *ACM* (PODC'08), 35-44.

Schneider, J. and Wattenhofer, R. (2010) A new technique for distributed symmetry breaking. *ACM* (PODC'10), 257-266.

Shi, Y. and Eberhart, R.C. (1998) A modified particle swarm optimizer. *Proceedings of IEEE International conference of Evolutionary Computation*, 69-73.

Soïensen, K. (2013) Metaheuristics-the metaphor exposed. *Intl. Trans. in Op. Res.*, 0:1-16.

Storn, R. and Price, K. (1997) Differential evolution - A simple and efficient heuristic for global optimization over continuous spaces. *Journal of Global Optimization*, 11(4):341-459.

Storn, R. (2013) Differential evolution. Retrieved from: http://www1.icsi.berkeley.edu/ storn/-code.html

Tikhonov, A.V. (1943) On the stability of inverse problems. *C.R. (Doklady) Acad. Sci. URSS (N.S.)*, 32:176-179

Timmins M. and Ashlock D. (2017) Network induction for epidemic profiles with a novel representation. *Biosystems*, 162:205-215.

Turing, A.M. (1950) Computing machinery and intelligence. *Mind* LIX, (238):433–460.

Turner, J.S. (1988) Almost all $k-$colorable graphs are easy to color. *J.Algorithms*, 9:63-82.

van Lint, J.H. and Wilson, R. M. (2001), *A Course in Combinatorics (2nd ed.)*. Cambridge University Press, ISBN 0-521-80340-3

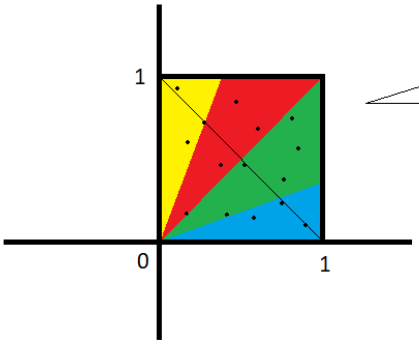
Welsh, B.J., and Powell, M.B. (1967) Ting routing and wavelength translation. *Prc. 9th Annual Symp. on Dis. Algorithms* (SODA '98), 33-341.

Zhang J., Avasarala, V., Sanderson, A.C., and Mullen, T., (2008) Differential evolution for discrete optimization: an experimental study on combinatorial auction problems. *Proc. IEEE World Congr. Comput. Itell.*, 2794-2800.

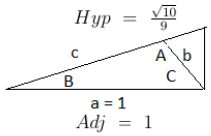
Appendix A

This appendix shows that taking a uniform random sample in both variable and then normalizing the resultant vector, using the 1 norm, will not give you a uniform random sample.

Calculating the area of the Triangles



Length of each portion of line equal $\sqrt{2}/4$



$Hyp = \frac{\sqrt{10}}{9}$

$Opp = \frac{1}{3}$

$Adj = 1$

Calculating the Area of the Blue Triangle

$C = \pi/4$ $\quad c^2 = a^2 + b^2 - 2ab * cos(C)$

$a = 1$ $\quad c^2 = (1)^2 + (\sqrt{2}/4)^2 - 2(1)(\sqrt{2}/4) * cos(\pi/4)$

$b = \sqrt{2}/4$ $\quad c^2 = 5/8$

$c = \sqrt{5/8}$

$B = \arccos(\frac{a^2+c^2-b^2}{2ac}) = arccos(Adj/Hyp)$

$\because (\frac{a^2+c^2-b^2}{2ac}) = \frac{Adj}{Hyp}$

$\because Adj = a = 1, \quad Hyp = (\frac{2ac}{a^2+c^2-b^2})$

$Hyp = \frac{2(\sqrt{5/8})}{(1)^2+(\sqrt{5/8})^2-(\sqrt{2}/4)^2}$

$Hyp = \frac{\sqrt{10}}{9}$

$Opp^2 = Hyp^2 - Adj^2$

$Opp^2 = (\frac{\sqrt{10}}{9})^2 - 1^2 = \frac{10}{9} - 1 = \frac{1}{9}$

$Opp = \frac{1}{3}$

$A_{BT} = (b * h)/2 = (1 * (1/3))/2$

$A_{BT} = 1/6$

Calculating the Area of the Green Triangle

$A_{GT} = (b * h)/2$

$b = 1 - (1/3) = 2/3$

$h = 1$

$A_{GT} = ((2/3) * (1))/2$

$A_{GT} = 1/3$

Calculating the Area of the Red Triangle

$A_{RT} = A_{GT}$

$A_{RT} = 1/3$

Calculating the Area of the Yellow Triangle

$A_{YT} = A_{BT}$

$A_{YT} = 1/6$

Appendix B

Results of Edit Command Testing

This appendix shows the parameter settings, includting the chromosomal regulatory sequences, that are used for each of the experiments that were designed to test the edit commands. The order that the different parameters appear are population size, number of mutations, number of iterations, amount of toggle, amount of hop, amount of add, amount of delete, amount of swap, amount of local toggle and amount of null.

B.1 Toggle

1000	5	1000000	0	0.3	0.3	0.2	0	0.2	0
1000	5	1000000	0.03	0.291	0.291	0.194	0	0.194	0
1000	5	1000000	0.06	0.282	0.282	0.188	0	0.188	0
1000	5	1000000	0.09	0.273	0.273	0.182	0	0.182	0
1000	5	1000000	0.12	0.264	0.264	0.176	0	0.176	0
1000	5	1000000	0.15	0.255	0.255	0.17	0	0.17	0
1000	5	1000000	0.18	0.246	0.246	0.164	0	0.164	0
1000	5	1000000	0.21	0.237	0.237	0.158	0	0.158	0
1000	5	1000000	0.24	0.228	0.228	0.152	0	0.152	0
1000	5	1000000	0.27	0.219	0.219	0.146	0	0.146	0
1000	5	1000000	0.3	0.21	0.21	0.14	0	0.14	0

(a) Run 1 Group 1

1000	5	1000000	0	0.21	0.21	0.14	0	0.14	0.3
1000	5	1000000	0.03	0.21	0.21	0.14	0	0.14	0.27
1000	5	1000000	0.06	0.21	0.21	0.14	0	0.14	0.24
1000	5	1000000	0.09	0.21	0.21	0.14	0	0.14	0.21
1000	5	1000000	0.12	0.21	0.21	0.14	0	0.14	0.18
1000	5	1000000	0.15	0.21	0.21	0.14	0	0.14	0.15
1000	5	1000000	0.18	0.21	0.21	0.14	0	0.14	0.12
1000	5	1000000	0.21	0.21	0.21	0.14	0	0.14	0.09
1000	5	1000000	0.24	0.21	0.21	0.14	0	0.14	0.06
1000	5	1000000	0.27	0.21	0.21	0.14	0	0.14	0.03
1000	5	1000000	0.3	0.21	0.21	0.14	0	0.14	0

(b) Run 2 Group 1

1000	5	1000000	0	0.45	0.25	0	0.3	0	0
1000	5	1000000	0.03	0.4365	0.2425	0	0.291	0	0
1000	5	1000000	0.06	0.423	0.235	0	0.282	0	0
1000	5	1000000	0.09	0.4095	0.2275	0	0.273	0	0
1000	5	1000000	0.12	0.396	0.22	0	0.264	0	0
1000	5	1000000	0.15	0.3825	0.2125	0	0.255	0	0
1000	5	1000000	0.18	0.369	0.205	0	0.246	0	0
1000	5	1000000	0.21	0.3555	0.1975	0	0.237	0	0
1000	5	1000000	0.24	0.342	0.19	0	0.228	0	0
1000	5	1000000	0.27	0.3285	0.1825	0	0.219	0	0
1000	5	1000000	0.3	0.315	0.175	0	0.21	0	0

(c) Run 1 Group 2

1000	5	1000000	0.3	0.21	0.21	0.14	0	0.14	0
1000	5	1000000	0	0.315	0.175	0	0.21	0	0.3
1000	5	1000000	0.03	0.315	0.175	0	0.21	0	0.27
1000	5	1000000	0.06	0.315	0.175	0	0.21	0	0.24
1000	5	1000000	0.09	0.315	0.175	0	0.21	0	0.21
1000	5	1000000	0.12	0.315	0.175	0	0.21	0	0.18
1000	5	1000000	0.15	0.315	0.175	0	0.21	0	0.15
1000	5	1000000	0.18	0.315	0.175	0	0.21	0	0.12
1000	5	1000000	0.21	0.315	0.175	0	0.21	0	0.09
1000	5	1000000	0.24	0.315	0.175	0	0.21	0	0.06
1000	5	1000000	0.27	0.315	0.175	0	0.21	0	0.03
1000	5	1000000	0.3	0.315	0.175	0	0.21	0	0

(d) Run 2 Group 2

1000	5	1000000	0	0.2	0.2	0.4	0	0.2	0
1000	5	1000000	0.03	0.194	0.194	0.388	0	0.194	0
1000	5	1000000	0.06	0.188	0.188	0.376	0	0.188	0
1000	5	1000000	0.09	0.182	0.182	0.364	0	0.182	0
1000	5	1000000	0.12	0.176	0.176	0.352	0	0.176	0
1000	5	1000000	0.15	0.17	0.17	0.34	0	0.17	0
1000	5	1000000	0.18	0.164	0.164	0.328	0	0.164	0
1000	5	1000000	0.21	0.158	0.158	0.316	0	0.158	0
1000	5	1000000	0.24	0.152	0.152	0.304	0	0.152	0
1000	5	1000000	0.27	0.146	0.146	0.292	0	0.146	0
1000	5	1000000	0.3	0.14	0.14	0.28	0	0.14	0

(e) Run 1 Group 3

1000	5	1000000	0	0.14	0.14	0.28	0	0.14	0.3
1000	5	1000000	0.03	0.14	0.14	0.28	0	0.14	0.27
1000	5	1000000	0.06	0.14	0.14	0.28	0	0.14	0.24
1000	5	1000000	0.09	0.14	0.14	0.28	0	0.14	0.21
1000	5	1000000	0.12	0.14	0.14	0.28	0	0.14	0.18
1000	5	1000000	0.15	0.14	0.14	0.28	0	0.14	0.15
1000	5	1000000	0.18	0.14	0.14	0.28	0	0.14	0.12
1000	5	1000000	0.21	0.14	0.14	0.28	0	0.14	0.09
1000	5	1000000	0.24	0.14	0.14	0.28	0	0.14	0.06
1000	5	1000000	0.27	0.14	0.14	0.28	0	0.14	0.03
1000	5	1000000	0.3	0.14	0.14	0.28	0	0.14	0

(f) Run 2 Group 3

1000	5	1000000	0	0.25	0.25	0.1	0	0.4	0
1000	5	1000000	0.03	0.2425	0.2425	0.097	0	0.388	0
1000	5	1000000	0.06	0.235	0.235	0.094	0	0.376	0
1000	5	1000000	0.09	0.2275	0.2275	0.091	0	0.364	0
1000	5	1000000	0.12	0.22	0.22	0.088	0	0.352	0
1000	5	1000000	0.15	0.2125	0.2125	0.085	0	0.34	0
1000	5	1000000	0.18	0.205	0.205	0.082	0	0.328	0
1000	5	1000000	0.21	0.1975	0.1975	0.079	0	0.316	0
1000	5	1000000	0.24	0.19	0.19	0.076	0	0.304	0
1000	5	1000000	0.27	0.1825	0.1825	0.073	0	0.292	0
1000	5	1000000	0.3	0.175	0.175	0.07	0	0.28	0

(g) Run 1 Group 4

1000	5	1000000	0	0.175	0.175	0.07	0	0.28	0.3
1000	5	1000000	0.03	0.175	0.175	0.07	0	0.28	0.27
1000	5	1000000	0.06	0.175	0.175	0.07	0	0.28	0.24
1000	5	1000000	0.09	0.175	0.175	0.07	0	0.28	0.21
1000	5	1000000	0.12	0.175	0.175	0.07	0	0.28	0.18
1000	5	1000000	0.15	0.175	0.175	0.07	0	0.28	0.15
1000	5	1000000	0.18	0.175	0.175	0.07	0	0.28	0.12
1000	5	1000000	0.21	0.175	0.175	0.07	0	0.28	0.09
1000	5	1000000	0.24	0.175	0.175	0.07	0	0.28	0.06
1000	5	1000000	0.27	0.175	0.175	0.07	0	0.28	0.03
1000	5	1000000	0.3	0.175	0.175	0.07	0	0.28	0

(h) Run 2 Group 4

1000	5	1000000	0	0.3	0.5	0.1	0	0.1	0
1000	5	1000000	0.03	0.291	0.485	0.097	0	0.097	0
1000	5	1000000	0.06	0.282	0.47	0.094	0	0.094	0
1000	5	1000000	0.09	0.273	0.455	0.091	0	0.091	0
1000	5	1000000	0.12	0.264	0.44	0.088	0	0.088	0
1000	5	1000000	0.15	0.255	0.425	0.085	0	0.085	0
1000	5	1000000	0.18	0.246	0.41	0.082	0	0.082	0
1000	5	1000000	0.21	0.237	0.395	0.079	0	0.079	0
1000	5	1000000	0.24	0.228	0.38	0.076	0	0.076	0
1000	5	1000000	0.27	0.219	0.365	0.073	0	0.073	0
1000	5	1000000	0.3	0.21	0.35	0.07	0	0.07	0

(i) Run 1 Group 5

1000	5	1000000	0	0.21	0.35	0.07	0	0.07	0.3
1000	5	1000000	0.03	0.21	0.35	0.07	0	0.07	0.27
1000	5	1000000	0.06	0.21	0.35	0.07	0	0.07	0.24
1000	5	1000000	0.09	0.21	0.35	0.07	0	0.07	0.21
1000	5	1000000	0.12	0.21	0.35	0.07	0	0.07	0.18
1000	5	1000000	0.15	0.21	0.35	0.07	0	0.07	0.15
1000	5	1000000	0.18	0.21	0.35	0.07	0	0.07	0.12
1000	5	1000000	0.21	0.21	0.35	0.07	0	0.07	0.09
1000	5	1000000	0.24	0.21	0.35	0.07	0	0.07	0.06
1000	5	1000000	0.27	0.21	0.35	0.07	0	0.07	0.03
1000	5	1000000	0.3	0.21	0.35	0.07	0	0.07	0

(j) Run 2 Group 5

1000	5	1000000	0	0.1	0.1	0.4	0	0.4	0
1000	5	1000000	0.03	0.097	0.097	0.388	0	0.388	0
1000	5	1000000	0.06	0.094	0.094	0.376	0	0.376	0
1000	5	1000000	0.09	0.091	0.091	0.364	0	0.364	0
1000	5	1000000	0.12	0.088	0.088	0.352	0	0.352	0
1000	5	1000000	0.15	0.085	0.085	0.34	0	0.34	0
1000	5	1000000	0.18	0.082	0.082	0.328	0	0.328	0
1000	5	1000000	0.21	0.079	0.079	0.316	0	0.316	0
1000	5	1000000	0.24	0.076	0.076	0.304	0	0.304	0
1000	5	1000000	0.27	0.073	0.073	0.292	0	0.292	0
1000	5	1000000	0.3	0.07	0.07	0.28	0	0.28	0

(k) Run 1 Group 6

1000	5	1000000	0	0.07	0.07	0.28	0	0.28	0.3
1000	5	1000000	0.03	0.07	0.07	0.28	0	0.28	0.27
1000	5	1000000	0.06	0.07	0.07	0.28	0	0.28	0.24
1000	5	1000000	0.09	0.07	0.07	0.28	0	0.28	0.21
1000	5	1000000	0.12	0.07	0.07	0.28	0	0.28	0.18
1000	5	1000000	0.15	0.07	0.07	0.28	0	0.28	0.15
1000	5	1000000	0.18	0.07	0.07	0.28	0	0.28	0.12
1000	5	1000000	0.21	0.07	0.07	0.28	0	0.28	0.09
1000	5	1000000	0.24	0.07	0.07	0.28	0	0.28	0.06
1000	5	1000000	0.27	0.07	0.07	0.28	0	0.28	0.03
1000	5	1000000	0.3	0.07	0.07	0.28	0	0.28	0

(l) Run 2 Group 6

1000	5	1000000	0	0.25	0.25	0.25	0	0.25	0
1000	5	1000000	0.03	0.2425	0.2425	0.2425	0	0.2425	0
1000	5	1000000	0.06	0.235	0.235	0.235	0	0.235	0
1000	5	1000000	0.09	0.2275	0.2275	0.2275	0	0.2275	0
1000	5	1000000	0.12	0.22	0.22	0.22	0	0.22	0
1000	5	1000000	0.15	0.2125	0.2125	0.2125	0	0.2125	0
1000	5	1000000	0.18	0.205	0.205	0.205	0	0.205	0
1000	5	1000000	0.21	0.1975	0.1975	0.1975	0	0.1975	0
1000	5	1000000	0.24	0.19	0.19	0.19	0	0.19	0
1000	5	1000000	0.27	0.1825	0.1825	0.1825	0	0.1825	0
1000	5	1000000	0.3	0.175	0.175	0.175	0	0.175	0

(m) Run 1 Group 7

1000	5	1000000	0	0.175	0.175	0.175	0	0.175	0.3
1000	5	1000000	0.03	0.175	0.175	0.175	0	0.175	0.27
1000	5	1000000	0.06	0.175	0.175	0.175	0	0.175	0.24
1000	5	1000000	0.09	0.175	0.175	0.175	0	0.175	0.21
1000	5	1000000	0.12	0.175	0.175	0.175	0	0.175	0.18
1000	5	1000000	0.15	0.175	0.175	0.175	0	0.175	0.15
1000	5	1000000	0.18	0.175	0.175	0.175	0	0.175	0.12
1000	5	1000000	0.21	0.175	0.175	0.175	0	0.175	0.09
1000	5	1000000	0.24	0.175	0.175	0.175	0	0.175	0.06
1000	5	1000000	0.27	0.175	0.175	0.175	0	0.175	0.03
1000	5	1000000	0.3	0.175	0.175	0.175	0	0.175	0

(n) Run 2 Group 7

1000	5	1000000	0	0.1	0.1	0.7	0	0.1	0
1000	5	1000000	0.03	0.097	0.097	0.679	0	0.097	0
1000	5	1000000	0.06	0.094	0.094	0.658	0	0.094	0
1000	5	1000000	0.09	0.091	0.091	0.637	0	0.091	0
1000	5	1000000	0.12	0.088	0.088	0.616	0	0.088	0
1000	5	1000000	0.15	0.085	0.085	0.595	0	0.085	0
1000	5	1000000	0.18	0.082	0.082	0.574	0	0.082	0
1000	5	1000000	0.21	0.079	0.079	0.553	0	0.079	0
1000	5	1000000	0.24	0.076	0.076	0.532	0	0.076	0
1000	5	1000000	0.27	0.073	0.073	0.511	0	0.073	0
1000	5	1000000	0.3	0.07	0.07	0.49	0	0.07	0

(o) Run 1 Group 8

1000	5	1000000	0	0.07	0.07	0.49	0	0.07	0.3
1000	5	1000000	0.03	0.07	0.07	0.49	0	0.07	0.27
1000	5	1000000	0.06	0.07	0.07	0.49	0	0.07	0.24
1000	5	1000000	0.09	0.07	0.07	0.49	0	0.07	0.21
1000	5	1000000	0.12	0.07	0.07	0.49	0	0.07	0.18
1000	5	1000000	0.15	0.07	0.07	0.49	0	0.07	0.15
1000	5	1000000	0.18	0.07	0.07	0.49	0	0.07	0.12
1000	5	1000000	0.21	0.07	0.07	0.49	0	0.07	0.09
1000	5	1000000	0.24	0.07	0.07	0.49	0	0.07	0.06
1000	5	1000000	0.27	0.07	0.07	0.49	0	0.07	0.03
1000	5	1000000	0.3	0.07	0.07	0.49	0	0.07	0

(p) Run 2 Group 8

1000	5	1000000	0	0.1	0.3	0.3	0	0.3	0
1000	5	1000000	0.03	0.097	0.291	0.291	0	0.291	0
1000	5	1000000	0.06	0.094	0.282	0.282	0	0.282	0
1000	5	1000000	0.09	0.091	0.273	0.273	0	0.273	0
1000	5	1000000	0.12	0.088	0.264	0.264	0	0.264	0
1000	5	1000000	0.15	0.085	0.255	0.255	0	0.255	0
1000	5	1000000	0.18	0.082	0.246	0.246	0	0.246	0
1000	5	1000000	0.21	0.079	0.237	0.237	0	0.237	0
1000	5	1000000	0.24	0.076	0.228	0.228	0	0.228	0
1000	5	1000000	0.27	0.073	0.219	0.219	0	0.219	0
1000	5	1000000	0.3	0.07	0.21	0.21	0	0.21	0

(q) Run 1 Group 9

1000	5	1000000	0	0.07	0.21	0.21	0	0.21	0.3
1000	5	1000000	0.03	0.07	0.21	0.21	0	0.21	0.27
1000	5	1000000	0.06	0.07	0.21	0.21	0	0.21	0.24
1000	5	1000000	0.09	0.07	0.21	0.21	0	0.21	0.21
1000	5	1000000	0.12	0.07	0.21	0.21	0	0.21	0.18
1000	5	1000000	0.15	0.07	0.21	0.21	0	0.21	0.15
1000	5	1000000	0.18	0.07	0.21	0.21	0	0.21	0.12
1000	5	1000000	0.21	0.07	0.21	0.21	0	0.21	0.09
1000	5	1000000	0.24	0.07	0.21	0.21	0	0.21	0.06
1000	5	1000000	0.27	0.07	0.21	0.21	0	0.21	0.03
1000	5	1000000	0.3	0.07	0.21	0.21	0	0.21	0

(r) Run 2 Group 9

1000	5	1000000	0	0.35	0.35	0.15	0	0.15	0
1000	5	1000000	0.03	0.3395	0.3395	0.1455	0	0.1455	0
1000	5	1000000	0.06	0.329	0.329	0.141	0	0.141	0
1000	5	1000000	0.09	0.3185	0.3185	0.1365	0	0.1365	0
1000	5	1000000	0.12	0.308	0.308	0.132	0	0.132	0
1000	5	1000000	0.15	0.2975	0.2975	0.1275	0	0.1275	0
1000	5	1000000	0.18	0.287	0.287	0.123	0	0.123	0
1000	5	1000000	0.21	0.2765	0.2765	0.1185	0	0.1185	0
1000	5	1000000	0.24	0.266	0.266	0.114	0	0.114	0
1000	5	1000000	0.27	0.2555	0.2555	0.1095	0	0.1095	0
1000	5	1000000	0.3	0.245	0.245	0.105	0	0.105	0

(s) Run 1 Group 10

1000	5	1000000	0	0.245	0.245	0.105	0	0.105	0.3
1000	5	1000000	0.03	0.245	0.245	0.105	0	0.105	0.27
1000	5	1000000	0.06	0.245	0.245	0.105	0	0.105	0.24
1000	5	1000000	0.09	0.245	0.245	0.105	0	0.105	0.21
1000	5	1000000	0.12	0.245	0.245	0.105	0	0.105	0.18
1000	5	1000000	0.15	0.245	0.245	0.105	0	0.105	0.15
1000	5	1000000	0.18	0.245	0.245	0.105	0	0.105	0.12
1000	5	1000000	0.21	0.245	0.245	0.105	0	0.105	0.09
1000	5	1000000	0.24	0.245	0.245	0.105	0	0.105	0.06
1000	5	1000000	0.27	0.245	0.245	0.105	0	0.105	0.03
1000	5	1000000	0.3	0.245	0.245	0.105	0	0.105	0

(t) Run 2 Group 10

1000	5	1000000	0.1	0	0	0	0	0	0.9
1000	5	1000000	0.2	0	0	0	0	0	0.8
1000	5	1000000	0.3	0	0	0	0	0	0.7
1000	5	1000000	0.4	0	0	0	0	0	0.6
1000	5	1000000	0.5	0	0	0	0	0	0.5
1000	5	1000000	0.6	0	0	0	0	0	0.4
1000	5	1000000	0.7	0	0	0	0	0	0.3
1000	5	1000000	0.8	0	0	0	0	0	0.2
1000	5	1000000	0.9	0	0	0	0	0	0.1
1000	5	1000000	1	0	0	0	0	0	0

(u) Null Experiment

Figure B.1: Local Toggle Experiments

B.2 Hop

1000	5	1000000	0.2	0	0.2	0.3	0	0.3	0
1000	5	1000000	0.194	0.03	0.194	0.291	0	0.291	0
1000	5	1000000	0.188	0.06	0.188	0.282	0	0.282	0
1000	5	1000000	0.182	0.09	0.182	0.273	0	0.273	0
1000	5	1000000	0.176	0.12	0.176	0.264	0	0.264	0
1000	5	1000000	0.17	0.15	0.17	0.255	0	0.255	0
1000	5	1000000	0.164	0.18	0.164	0.246	0	0.246	0
1000	5	1000000	0.158	0.21	0.158	0.237	0	0.237	0
1000	5	1000000	0.152	0.24	0.152	0.228	0	0.228	0
1000	5	1000000	0.146	0.27	0.146	0.219	0	0.219	0
1000	5	1000000	0.14	0.3	0.14	0.21	0	0.21	0

(a) Run 1 Group 1

1000	5	1000000	0.14	0	0.14	0.21	0	0.21	0.3
1000	5	1000000	0.14	0.03	0.14	0.21	0	0.21	0.27
1000	5	1000000	0.14	0.06	0.14	0.21	0	0.21	0.24
1000	5	1000000	0.14	0.09	0.14	0.21	0	0.21	0.21
1000	5	1000000	0.14	0.12	0.14	0.21	0	0.21	0.18
1000	5	1000000	0.14	0.15	0.14	0.21	0	0.21	0.15
1000	5	1000000	0.14	0.18	0.14	0.21	0	0.21	0.12
1000	5	1000000	0.14	0.21	0.14	0.21	0	0.21	0.09
1000	5	1000000	0.14	0.24	0.14	0.21	0	0.21	0.06
1000	5	1000000	0.14	0.27	0.14	0.21	0	0.21	0.03
1000	5	1000000	0.14	0.3	0.14	0.21	0	0.21	0

(b) Run 2 Group 1

1000	5	1000000	0.45	0	0.25	0	0	0.3	0
1000	5	1000000	0.4365	0.03	0.2425	0	0	0.291	0
1000	5	1000000	0.423	0.06	0.235	0	0	0.282	0
1000	5	1000000	0.4095	0.09	0.2275	0	0	0.273	0
1000	5	1000000	0.396	0.12	0.22	0	0	0.264	0
1000	5	1000000	0.3825	0.15	0.2125	0	0	0.255	0
1000	5	1000000	0.369	0.18	0.205	0	0	0.246	0
1000	5	1000000	0.3555	0.21	0.1975	0	0	0.237	0
1000	5	1000000	0.342	0.24	0.19	0	0	0.228	0
1000	5	1000000	0.3285	0.27	0.1825	0	0	0.219	0
1000	5	1000000	0.315	0.3	0.175	0	0	0.21	0

(c) Run 1 Group 2

1000	5	1000000	0.315	0	0.175	0	0	0.21	0.3
1000	5	1000000	0.315	0.03	0.175	0	0	0.21	0.27
1000	5	1000000	0.315	0.06	0.175	0	0	0.21	0.24
1000	5	1000000	0.315	0.09	0.175	0	0	0.21	0.21
1000	5	1000000	0.315	0.12	0.175	0	0	0.21	0.18
1000	5	1000000	0.315	0.15	0.175	0	0	0.21	0.15
1000	5	1000000	0.315	0.18	0.175	0	0	0.21	0.12
1000	5	1000000	0.315	0.21	0.175	0	0	0.21	0.09
1000	5	1000000	0.315	0.24	0.175	0	0	0.21	0.06
1000	5	1000000	0.315	0.27	0.175	0	0	0.21	0.03
1000	5	1000000	0.315	0.3	0.175	0	0	0.21	0

(d) Run 2 Group 2

1000	5	1000000	0.2	0	0.2	0.4	0	0.2	0
1000	5	1000000	0.194	0.03	0.194	0.388	0	0.194	0
1000	5	1000000	0.188	0.06	0.188	0.376	0	0.188	0
1000	5	1000000	0.182	0.09	0.182	0.364	0	0.182	0
1000	5	1000000	0.176	0.12	0.176	0.352	0	0.176	0
1000	5	1000000	0.17	0.15	0.17	0.34	0	0.17	0
1000	5	1000000	0.164	0.18	0.164	0.328	0	0.164	0
1000	5	1000000	0.158	0.21	0.158	0.316	0	0.158	0
1000	5	1000000	0.152	0.24	0.152	0.304	0	0.152	0
1000	5	1000000	0.146	0.27	0.146	0.292	0	0.146	0
1000	5	1000000	0.14	0.3	0.14	0.28	0	0.14	0

(e) Run 1 Group 3

1000	5	1000000	0.14	0	0.14	0.28	0	0.14	0.3
1000	5	1000000	0.14	0.03	0.14	0.28	0	0.14	0.27
1000	5	1000000	0.14	0.06	0.14	0.28	0	0.14	0.24
1000	5	1000000	0.14	0.09	0.14	0.28	0	0.14	0.21
1000	5	1000000	0.14	0.12	0.14	0.28	0	0.14	0.18
1000	5	1000000	0.14	0.15	0.14	0.28	0	0.14	0.15
1000	5	1000000	0.14	0.18	0.14	0.28	0	0.14	0.12
1000	5	1000000	0.14	0.21	0.14	0.28	0	0.14	0.09
1000	5	1000000	0.14	0.24	0.14	0.28	0	0.14	0.06
1000	5	1000000	0.14	0.27	0.14	0.28	0	0.14	0.03
1000	5	1000000	0.14	0.3	0.14	0.28	0	0.14	0

(f) Run 2 Group 3

1000	5	1000000	0.14	0.3	0.14	0.28	0	0.14	0
1000	5	1000000	0.3	0	0.3	0.1	0	0.3	0
1000	5	1000000	0.291	0.03	0.291	0.097	0	0.291	0
1000	5	1000000	0.282	0.06	0.282	0.094	0	0.282	0
1000	5	1000000	0.273	0.09	0.273	0.091	0	0.273	0
1000	5	1000000	0.264	0.12	0.264	0.088	0	0.264	0
1000	5	1000000	0.255	0.15	0.255	0.085	0	0.255	0
1000	5	1000000	0.246	0.18	0.246	0.082	0	0.246	0
1000	5	1000000	0.237	0.21	0.237	0.079	0	0.237	0
1000	5	1000000	0.228	0.24	0.228	0.076	0	0.228	0
1000	5	1000000	0.219	0.27	0.219	0.073	0	0.219	0
1000	5	1000000	0.21	0.3	0.21	0.07	0	0.21	0

(g) Run 1 Group 4

1000	5	1000000	0.21	0	0.21	0.07	0	0.21	0.3
1000	5	1000000	0.21	0.03	0.21	0.07	0	0.21	0.27
1000	5	1000000	0.21	0.06	0.21	0.07	0	0.21	0.24
1000	5	1000000	0.21	0.09	0.21	0.07	0	0.21	0.21
1000	5	1000000	0.21	0.12	0.21	0.07	0	0.21	0.18
1000	5	1000000	0.21	0.15	0.21	0.07	0	0.21	0.15
1000	5	1000000	0.21	0.18	0.21	0.07	0	0.21	0.12
1000	5	1000000	0.21	0.21	0.21	0.07	0	0.21	0.09
1000	5	1000000	0.21	0.24	0.21	0.07	0	0.21	0.06
1000	5	1000000	0.21	0.27	0.21	0.07	0	0.21	0.03
1000	5	1000000	0.21	0.3	0.21	0.07	0	0.21	0

(h) Run 2 Group 4

1000	5	1000000	0.4	0	0.3	0.15	0	0.15	0
1000	5	1000000	0.388	0.03	0.291	0.1455	0	0.1455	0
1000	5	1000000	0.376	0.06	0.282	0.141	0	0.141	0
1000	5	1000000	0.364	0.09	0.273	0.1365	0	0.1365	0
1000	5	1000000	0.352	0.12	0.264	0.132	0	0.132	0
1000	5	1000000	0.34	0.15	0.255	0.1275	0	0.1275	0
1000	5	1000000	0.328	0.18	0.246	0.123	0	0.123	0
1000	5	1000000	0.316	0.21	0.237	0.1185	0	0.1185	0
1000	5	1000000	0.304	0.24	0.228	0.114	0	0.114	0
1000	5	1000000	0.292	0.27	0.219	0.1095	0	0.1095	0
1000	5	1000000	0.28	0.3	0.21	0.105	0	0.105	0

(i) Run 1 Group 5

1000	5	1000000	0.28	0	0.21	0.105	0	0.105	0.3
1000	5	1000000	0.28	0.03	0.21	0.105	0	0.105	0.27
1000	5	1000000	0.28	0.06	0.21	0.105	0	0.105	0.24
1000	5	1000000	0.28	0.09	0.21	0.105	0	0.105	0.21
1000	5	1000000	0.28	0.12	0.21	0.105	0	0.105	0.18
1000	5	1000000	0.28	0.15	0.21	0.105	0	0.105	0.15
1000	5	1000000	0.28	0.18	0.21	0.105	0	0.105	0.12
1000	5	1000000	0.28	0.21	0.21	0.105	0	0.105	0.09
1000	5	1000000	0.28	0.24	0.21	0.105	0	0.105	0.06
1000	5	1000000	0.28	0.27	0.21	0.105	0	0.105	0.03
1000	5	1000000	0.28	0.3	0.21	0.105	0	0.105	0

(j) Run 2 Group 5

1000	5	1000000	0.2	0	0.6	0.1	0	0.1	0
1000	5	1000000	0.194	0.03	0.582	0.097	0	0.097	0
1000	5	1000000	0.188	0.06	0.564	0.094	0	0.094	0
1000	5	1000000	0.182	0.09	0.546	0.091	0	0.091	0
1000	5	1000000	0.176	0.12	0.528	0.088	0	0.088	0
1000	5	1000000	0.17	0.15	0.51	0.085	0	0.085	0
1000	5	1000000	0.164	0.18	0.492	0.082	0	0.082	0
1000	5	1000000	0.158	0.21	0.474	0.079	0	0.079	0
1000	5	1000000	0.152	0.24	0.456	0.076	0	0.076	0
1000	5	1000000	0.146	0.27	0.438	0.073	0	0.073	0
1000	5	1000000	0.14	0.3	0.42	0.07	0	0.07	0

(k) Run 1 Group 6

1000	5	1000000	0.14	0	0.42	0.07	0	0.07	0.3
1000	5	1000000	0.14	0.03	0.42	0.07	0	0.07	0.27
1000	5	1000000	0.14	0.06	0.42	0.07	0	0.07	0.24
1000	5	1000000	0.14	0.09	0.42	0.07	0	0.07	0.21
1000	5	1000000	0.14	0.12	0.42	0.07	0	0.07	0.18
1000	5	1000000	0.14	0.15	0.42	0.07	0	0.07	0.15
1000	5	1000000	0.14	0.18	0.42	0.07	0	0.07	0.12
1000	5	1000000	0.14	0.21	0.42	0.07	0	0.07	0.09
1000	5	1000000	0.14	0.24	0.42	0.07	0	0.07	0.06
1000	5	1000000	0.14	0.27	0.42	0.07	0	0.07	0.03
1000	5	1000000	0.14	0.3	0.42	0.07	0	0.07	0

(l) Run 2 Group 6

1000	5	1000000	0.2	0	0.4	0.2	0	0.2	0
1000	5	1000000	0.194	0.03	0.388	0.194	0	0.194	0
1000	5	1000000	0.188	0.06	0.376	0.188	0	0.188	0
1000	5	1000000	0.182	0.09	0.364	0.182	0	0.182	0
1000	5	1000000	0.176	0.12	0.352	0.176	0	0.176	0
1000	5	1000000	0.17	0.15	0.34	0.17	0	0.17	0
1000	5	1000000	0.164	0.18	0.328	0.164	0	0.164	0
1000	5	1000000	0.158	0.21	0.316	0.158	0	0.158	0
1000	5	1000000	0.152	0.24	0.304	0.152	0	0.152	0
1000	5	1000000	0.146	0.27	0.292	0.146	0	0.146	0
1000	5	1000000	0.14	0.3	0.28	0.14	0	0.14	0

(m) Run 1 Group 7

1000	5	1000000	0.14	0	0.28	0.14	0	0.14	0.3
1000	5	1000000	0.14	0.03	0.28	0.14	0	0.14	0.27
1000	5	1000000	0.14	0.06	0.28	0.14	0	0.14	0.24
1000	5	1000000	0.14	0.09	0.28	0.14	0	0.14	0.21
1000	5	1000000	0.14	0.12	0.28	0.14	0	0.14	0.18
1000	5	1000000	0.14	0.15	0.28	0.14	0	0.14	0.15
1000	5	1000000	0.14	0.18	0.28	0.14	0	0.14	0.12
1000	5	1000000	0.14	0.21	0.28	0.14	0	0.14	0.09
1000	5	1000000	0.14	0.24	0.28	0.14	0	0.14	0.06
1000	5	1000000	0.14	0.27	0.28	0.14	0	0.14	0.03
1000	5	1000000	0.14	0.3	0.28	0.14	0	0.14	0

(n) Run 2 Group 7

1000	5	1000000	0.1	0	0.1	0.7	0	0.1	0
1000	5	1000000	0.097	0.03	0.097	0.679	0	0.097	0
1000	5	1000000	0.094	0.06	0.094	0.658	0	0.094	0
1000	5	1000000	0.091	0.09	0.091	0.637	0	0.091	0
1000	5	1000000	0.088	0.12	0.088	0.616	0	0.088	0
1000	5	1000000	0.085	0.15	0.085	0.595	0	0.085	0
1000	5	1000000	0.082	0.18	0.082	0.574	0	0.082	0
1000	5	1000000	0.079	0.21	0.079	0.553	0	0.079	0
1000	5	1000000	0.076	0.24	0.076	0.532	0	0.076	0
1000	5	1000000	0.073	0.27	0.073	0.511	0	0.073	0
1000	5	1000000	0.07	0.3	0.07	0.49	0	0.07	0

(o) Run 1 Group 8

1000	5	1000000	0.07	0	0.07	0.49	0	0.07	0.3
1000	5	1000000	0.07	0.03	0.07	0.49	0	0.07	0.27
1000	5	1000000	0.07	0.06	0.07	0.49	0	0.07	0.24
1000	5	1000000	0.07	0.09	0.07	0.49	0	0.07	0.21
1000	5	1000000	0.07	0.12	0.07	0.49	0	0.07	0.18
1000	5	1000000	0.07	0.15	0.07	0.49	0	0.07	0.15
1000	5	1000000	0.07	0.18	0.07	0.49	0	0.07	0.12
1000	5	1000000	0.07	0.21	0.07	0.49	0	0.07	0.09
1000	5	1000000	0.07	0.24	0.07	0.49	0	0.07	0.06
1000	5	1000000	0.07	0.27	0.07	0.49	0	0.07	0.03
1000	5	1000000	0.07	0.3	0.07	0.49	0	0.07	0

(p) Run 2 Group 8

1000	5	1000000	0.25	0	0.25	0.25	0	0.25	0
1000	5	1000000	0.2425	0.03	0.2425	0.2425	0	0.2425	0
1000	5	1000000	0.235	0.06	0.235	0.235	0	0.235	0
1000	5	1000000	0.2275	0.09	0.2275	0.2275	0	0.2275	0
1000	5	1000000	0.22	0.12	0.22	0.22	0	0.22	0
1000	5	1000000	0.2125	0.15	0.2125	0.2125	0	0.2125	0
1000	5	1000000	0.205	0.18	0.205	0.205	0	0.205	0
1000	5	1000000	0.1975	0.21	0.1975	0.1975	0	0.1975	0
1000	5	1000000	0.19	0.24	0.19	0.19	0	0.19	0
1000	5	1000000	0.1825	0.27	0.1825	0.1825	0	0.1825	0
1000	5	1000000	0.175	0.3	0.175	0.175	0	0.175	0

(q) Run 1 Group 9

1000	5	1000000	0.175	0	0.175	0.175	0	0.175	0.3
1000	5	1000000	0.175	0.03	0.175	0.175	0	0.175	0.27
1000	5	1000000	0.175	0.06	0.175	0.175	0	0.175	0.24
1000	5	1000000	0.175	0.09	0.175	0.175	0	0.175	0.21
1000	5	1000000	0.175	0.12	0.175	0.175	0	0.175	0.18
1000	5	1000000	0.175	0.15	0.175	0.175	0	0.175	0.15
1000	5	1000000	0.175	0.18	0.175	0.175	0	0.175	0.12
1000	5	1000000	0.175	0.21	0.175	0.175	0	0.175	0.09
1000	5	1000000	0.175	0.24	0.175	0.175	0	0.175	0.06
1000	5	1000000	0.175	0.27	0.175	0.175	0	0.175	0.03
1000	5	1000000	0.175	0.3	0.175	0.175	0	0.175	0

(r) Run 2 Group 9

1000	5	1000000	0.4	0	0.4	0.1	0	0.1	0
1000	5	1000000	0.388	0.03	0.388	0.097	0	0.097	0
1000	5	1000000	0.376	0.06	0.376	0.094	0	0.094	0
1000	5	1000000	0.364	0.09	0.364	0.091	0	0.091	0
1000	5	1000000	0.352	0.12	0.352	0.088	0	0.088	0
1000	5	1000000	0.34	0.15	0.34	0.085	0	0.085	0
1000	5	1000000	0.328	0.18	0.328	0.082	0	0.082	0
1000	5	1000000	0.316	0.21	0.316	0.079	0	0.079	0
1000	5	1000000	0.304	0.24	0.304	0.076	0	0.076	0
1000	5	1000000	0.292	0.27	0.292	0.073	0	0.073	0
1000	5	1000000	0.28	0.3	0.28	0.07	0	0.07	0

(s) Run 1 Group 10

1000	5	1000000	0.28	0	0.28	0.07	0	0.07	0.3
1000	5	1000000	0.28	0.03	0.28	0.07	0	0.07	0.27
1000	5	1000000	0.28	0.06	0.28	0.07	0	0.07	0.24
1000	5	1000000	0.28	0.09	0.28	0.07	0	0.07	0.21
1000	5	1000000	0.28	0.12	0.28	0.07	0	0.07	0.18
1000	5	1000000	0.28	0.15	0.28	0.07	0	0.07	0.15
1000	5	1000000	0.28	0.18	0.28	0.07	0	0.07	0.12
1000	5	1000000	0.28	0.21	0.28	0.07	0	0.07	0.09
1000	5	1000000	0.28	0.24	0.28	0.07	0	0.07	0.06
1000	5	1000000	0.28	0.27	0.28	0.07	0	0.07	0.03
1000	5	1000000	0.28	0.3	0.28	0.07	0	0.07	0

(t) Run 2 Group 10

1000	5	1000000	0	0.1	0	0	0	0	0.9
1000	5	1000000	0	0.2	0	0	0	0	0.8
1000	5	1000000	0	0.3	0	0	0	0	0.7
1000	5	1000000	0	0.4	0	0	0	0	0.6
1000	5	1000000	0	0.5	0	0	0	0	0.5
1000	5	1000000	0	0.6	0	0	0	0	0.4
1000	5	1000000	0	0.7	0	0	0	0	0.3
1000	5	1000000	0	0.8	0	0	0	0	0.2
1000	5	1000000	0	0.9	0	0	0	0	0.1
1000	5	1000000	0	1	0	0	0	0	0

(u) Null Experiment

Figure B.2: Hop's Experimental Settings

B.3 Add

1000	5	1000000	0.2	0.2	0	0.3	0	0.3	0
1000	5	1000000	0.194	0.194	0.03	0.291	0	0.291	0
1000	5	1000000	0.188	0.188	0.06	0.282	0	0.282	0
1000	5	1000000	0.182	0.182	0.09	0.273	0	0.273	0
1000	5	1000000	0.176	0.176	0.12	0.264	0	0.264	0
1000	5	1000000	0.17	0.17	0.15	0.255	0	0.255	0
1000	5	1000000	0.164	0.164	0.18	0.246	0	0.246	0
1000	5	1000000	0.158	0.158	0.21	0.237	0	0.237	0
1000	5	1000000	0.152	0.152	0.24	0.228	0	0.228	0
1000	5	1000000	0.146	0.146	0.27	0.219	0	0.219	0
1000	5	1000000	0.14	0.14	0.3	0.21	0	0.21	0

(a) Run 1 Group 1

1000	5	1000000	0.14	0.14	0	0.21	0	0.21	0.3
1000	5	1000000	0.14	0.14	0.03	0.21	0	0.21	0.27
1000	5	1000000	0.14	0.14	0.06	0.21	0	0.21	0.24
1000	5	1000000	0.14	0.14	0.09	0.21	0	0.21	0.21
1000	5	1000000	0.14	0.14	0.12	0.21	0	0.21	0.18
1000	5	1000000	0.14	0.14	0.15	0.21	0	0.21	0.15
1000	5	1000000	0.14	0.14	0.18	0.21	0	0.21	0.12
1000	5	1000000	0.14	0.14	0.21	0.21	0	0.21	0.09
1000	5	1000000	0.14	0.14	0.24	0.21	0	0.21	0.06
1000	5	1000000	0.14	0.14	0.27	0.21	0	0.21	0.03
1000	5	1000000	0.14	0.14	0.3	0.21	0	0.21	0

(b) Run 2 Group 1

1000	5	1000000	0.45	0.25	0	0	0	0.3	0
1000	5	1000000	0.4365	0.2425	0.03	0	0	0.291	0
1000	5	1000000	0.423	0.235	0.06	0	0	0.282	0
1000	5	1000000	0.4095	0.2275	0.09	0	0	0.273	0
1000	5	1000000	0.396	0.22	0.12	0	0	0.264	0
1000	5	1000000	0.3825	0.2125	0.15	0	0	0.255	0
1000	5	1000000	0.369	0.205	0.18	0	0	0.246	0
1000	5	1000000	0.3555	0.1975	0.21	0	0	0.237	0
1000	5	1000000	0.342	0.19	0.24	0	0	0.228	0
1000	5	1000000	0.3285	0.1825	0.27	0	0	0.219	0
1000	5	1000000	0.315	0.175	0.3	0	0	0.21	0

(c) Run 1 Group 2

1000	5	1000000	0.315	0.175	0	0	0	0.21	0.3
1000	5	1000000	0.315	0.175	0.03	0	0	0.21	0.27
1000	5	1000000	0.315	0.175	0.06	0	0	0.21	0.24
1000	5	1000000	0.315	0.175	0.09	0	0	0.21	0.21
1000	5	1000000	0.315	0.175	0.12	0	0	0.21	0.18
1000	5	1000000	0.315	0.175	0.15	0	0	0.21	0.15
1000	5	1000000	0.315	0.175	0.18	0	0	0.21	0.12
1000	5	1000000	0.315	0.175	0.21	0	0	0.21	0.09
1000	5	1000000	0.315	0.175	0.24	0	0	0.21	0.06
1000	5	1000000	0.315	0.175	0.27	0	0	0.21	0.03
1000	5	1000000	0.315	0.175	0.3	0	0	0.21	0

(d) Run 2 Group 2

1000	5	1000000	0.2	0.2	0	0.4	0	0.2	0
1000	5	1000000	0.194	0.194	0.03	0.388	0	0.194	0
1000	5	1000000	0.188	0.188	0.06	0.376	0	0.188	0
1000	5	1000000	0.182	0.182	0.09	0.364	0	0.182	0
1000	5	1000000	0.176	0.176	0.12	0.352	0	0.176	0
1000	5	1000000	0.17	0.17	0.15	0.34	0	0.17	0
1000	5	1000000	0.164	0.164	0.18	0.328	0	0.164	0
1000	5	1000000	0.158	0.158	0.21	0.316	0	0.158	0
1000	5	1000000	0.152	0.152	0.24	0.304	0	0.152	0
1000	5	1000000	0.146	0.146	0.27	0.292	0	0.146	0
1000	5	1000000	0.14	0.14	0.3	0.28	0	0.14	0

(e) Run 1 Group 3

1000	5	1000000	0.14	0.14	0	0.28	0	0.14	0.3
1000	5	1000000	0.14	0.14	0.03	0.28	0	0.14	0.27
1000	5	1000000	0.14	0.14	0.06	0.28	0	0.14	0.24
1000	5	1000000	0.14	0.14	0.09	0.28	0	0.14	0.21
1000	5	1000000	0.14	0.14	0.12	0.28	0	0.14	0.18
1000	5	1000000	0.14	0.14	0.15	0.28	0	0.14	0.15
1000	5	1000000	0.14	0.14	0.18	0.28	0	0.14	0.12
1000	5	1000000	0.14	0.14	0.21	0.28	0	0.14	0.09
1000	5	1000000	0.14	0.14	0.24	0.28	0	0.14	0.06
1000	5	1000000	0.14	0.14	0.27	0.28	0	0.14	0.03
1000	5	1000000	0.14	0.14	0.3	0.28	0	0.14	0

(f) Run 2 Group 3

1000	5	1000000	0.3	0.3	0	0.1	0	0.3	0
1000	5	1000000	0.291	0.291	0.03	0.097	0	0.291	0
1000	5	1000000	0.282	0.282	0.06	0.094	0	0.282	0
1000	5	1000000	0.273	0.273	0.09	0.091	0	0.273	0
1000	5	1000000	0.264	0.264	0.12	0.088	0	0.264	0
1000	5	1000000	0.255	0.255	0.15	0.085	0	0.255	0
1000	5	1000000	0.246	0.246	0.18	0.082	0	0.246	0
1000	5	1000000	0.237	0.237	0.21	0.079	0	0.237	0
1000	5	1000000	0.228	0.228	0.24	0.076	0	0.228	0
1000	5	1000000	0.219	0.219	0.27	0.073	0	0.219	0
1000	5	1000000	0.21	0.21	0.3	0.07	0	0.21	0

(g) Run 1 Group 4

1000	5	1000000	0.21	0.21	0	0.07	0	0.21	0.3
1000	5	1000000	0.21	0.21	0.03	0.07	0	0.21	0.27
1000	5	1000000	0.21	0.21	0.06	0.07	0	0.21	0.24
1000	5	1000000	0.21	0.21	0.09	0.07	0	0.21	0.21
1000	5	1000000	0.21	0.21	0.12	0.07	0	0.21	0.18
1000	5	1000000	0.21	0.21	0.15	0.07	0	0.21	0.15
1000	5	1000000	0.21	0.21	0.18	0.07	0	0.21	0.12
1000	5	1000000	0.21	0.21	0.21	0.07	0	0.21	0.09
1000	5	1000000	0.21	0.21	0.24	0.07	0	0.21	0.06
1000	5	1000000	0.21	0.21	0.27	0.07	0	0.21	0.03
1000	5	1000000	0.21	0.21	0.3	0.07	0	0.21	0

(h) Run 2 Group 4

1000	5	1000000	0.4	0.3	0	0.15	0	0.15	0
1000	5	1000000	0.388	0.291	0.03	0.1455	0	0.1455	0
1000	5	1000000	0.376	0.282	0.06	0.141	0	0.141	0
1000	5	1000000	0.364	0.273	0.09	0.1365	0	0.1365	0
1000	5	1000000	0.352	0.264	0.12	0.132	0	0.132	0
1000	5	1000000	0.34	0.255	0.15	0.1275	0	0.1275	0
1000	5	1000000	0.328	0.246	0.18	0.123	0	0.123	0
1000	5	1000000	0.316	0.237	0.21	0.1185	0	0.1185	0
1000	5	1000000	0.304	0.228	0.24	0.114	0	0.114	0
1000	5	1000000	0.292	0.219	0.27	0.1095	0	0.1095	0
1000	5	1000000	0.28	0.21	0.3	0.105	0	0.105	0

(i) Run 1 Group 5

1000	5	1000000	0.28	0.21	0	0.105	0	0.105	0.3
1000	5	1000000	0.28	0.21	0.03	0.105	0	0.105	0.27
1000	5	1000000	0.28	0.21	0.06	0.105	0	0.105	0.24
1000	5	1000000	0.28	0.21	0.09	0.105	0	0.105	0.21
1000	5	1000000	0.28	0.21	0.12	0.105	0	0.105	0.18
1000	5	1000000	0.28	0.21	0.15	0.105	0	0.105	0.15
1000	5	1000000	0.28	0.21	0.18	0.105	0	0.105	0.12
1000	5	1000000	0.28	0.21	0.21	0.105	0	0.105	0.09
1000	5	1000000	0.28	0.21	0.24	0.105	0	0.105	0.06
1000	5	1000000	0.28	0.21	0.27	0.105	0	0.105	0.03
1000	5	1000000	0.28	0.21	0.3	0.105	0	0.105	0

(j) Run 2 Group 5

1000	5	1000000	0.2	0.6	0	0.1	0	0.1	0
1000	5	1000000	0.194	0.582	0.03	0.097	0	0.097	0
1000	5	1000000	0.188	0.564	0.06	0.094	0	0.094	0
1000	5	1000000	0.182	0.546	0.09	0.091	0	0.091	0
1000	5	1000000	0.176	0.528	0.12	0.088	0	0.088	0
1000	5	1000000	0.17	0.51	0.15	0.085	0	0.085	0
1000	5	1000000	0.164	0.492	0.18	0.082	0	0.082	0
1000	5	1000000	0.158	0.474	0.21	0.079	0	0.079	0
1000	5	1000000	0.152	0.456	0.24	0.076	0	0.076	0
1000	5	1000000	0.146	0.438	0.27	0.073	0	0.073	0
1000	5	1000000	0.28	0.21	0.3	0.105	0	0.105	0

(k) Run 1 Group 6

1000	5	1000000	0.14	0.42	0	0.07	0	0.07	0.3
1000	5	1000000	0.14	0.42	0.03	0.07	0	0.07	0.27
1000	5	1000000	0.14	0.42	0.06	0.07	0	0.07	0.24
1000	5	1000000	0.14	0.42	0.09	0.07	0	0.07	0.21
1000	5	1000000	0.14	0.42	0.12	0.07	0	0.07	0.18
1000	5	1000000	0.14	0.42	0.15	0.07	0	0.07	0.15
1000	5	1000000	0.14	0.42	0.18	0.07	0	0.07	0.12
1000	5	1000000	0.14	0.42	0.21	0.07	0	0.07	0.09
1000	5	1000000	0.14	0.42	0.24	0.07	0	0.07	0.06
1000	5	1000000	0.14	0.42	0.27	0.07	0	0.07	0.03
1000	5	1000000	0.14	0.42	0.3	0.07	0	0.07	0

(l) Run 2 Group 6

1000	5	1000000	0.2	0.4	0	0.2	0	0.2	0
1000	5	1000000	0.194	0.388	0.03	0.194	0	0.194	0
1000	5	1000000	0.188	0.376	0.06	0.188	0	0.188	0
1000	5	1000000	0.182	0.364	0.09	0.182	0	0.182	0
1000	5	1000000	0.176	0.352	0.12	0.176	0	0.176	0
1000	5	1000000	0.17	0.34	0.15	0.17	0	0.17	0
1000	5	1000000	0.164	0.328	0.18	0.164	0	0.164	0
1000	5	1000000	0.158	0.316	0.21	0.158	0	0.158	0
1000	5	1000000	0.152	0.304	0.24	0.152	0	0.152	0
1000	5	1000000	0.146	0.292	0.27	0.146	0	0.146	0
1000	5	1000000	0.14	0.28	0.3	0.14	0	0.14	0

(m) Run 1 Group 7

1000	5	1000000	0.14	0.28	0	0.14	0	0.14	0.3
1000	5	1000000	0.14	0.28	0.03	0.14	0	0.14	0.27
1000	5	1000000	0.14	0.28	0.06	0.14	0	0.14	0.24
1000	5	1000000	0.14	0.28	0.09	0.14	0	0.14	0.21
1000	5	1000000	0.14	0.28	0.12	0.14	0	0.14	0.18
1000	5	1000000	0.14	0.28	0.15	0.14	0	0.14	0.15
1000	5	1000000	0.14	0.28	0.18	0.14	0	0.14	0.12
1000	5	1000000	0.14	0.28	0.21	0.14	0	0.14	0.09
1000	5	1000000	0.14	0.28	0.24	0.14	0	0.14	0.06
1000	5	1000000	0.14	0.28	0.27	0.14	0	0.14	0.03
1000	5	1000000	0.14	0.28	0.3	0.14	0	0.14	0

(n) Run 2 Group 7

1000	5	1000000	0.1	0.1	0	0.7	0	0.1	0
1000	5	1000000	0.097	0.097	0.03	0.679	0	0.097	0
1000	5	1000000	0.094	0.094	0.06	0.658	0	0.094	0
1000	5	1000000	0.091	0.091	0.09	0.637	0	0.091	0
1000	5	1000000	0.088	0.088	0.12	0.616	0	0.088	0
1000	5	1000000	0.085	0.085	0.15	0.595	0	0.085	0
1000	5	1000000	0.082	0.082	0.18	0.574	0	0.082	0
1000	5	1000000	0.079	0.079	0.21	0.553	0	0.079	0
1000	5	1000000	0.076	0.076	0.24	0.532	0	0.076	0
1000	5	1000000	0.073	0.073	0.27	0.511	0	0.073	0
1000	5	1000000	0.07	0.07	0.3	0.49	0	0.07	0

(o) Run 1 Group 8

1000	5	1000000	0.07	0.07	0	0.49	0	0.07	0.3
1000	5	1000000	0.07	0.07	0.03	0.49	0	0.07	0.27
1000	5	1000000	0.07	0.07	0.06	0.49	0	0.07	0.24
1000	5	1000000	0.07	0.07	0.09	0.49	0	0.07	0.21
1000	5	1000000	0.07	0.07	0.12	0.49	0	0.07	0.18
1000	5	1000000	0.07	0.07	0.15	0.49	0	0.07	0.15
1000	5	1000000	0.07	0.07	0.18	0.49	0	0.07	0.12
1000	5	1000000	0.07	0.07	0.21	0.49	0	0.07	0.09
1000	5	1000000	0.07	0.07	0.24	0.49	0	0.07	0.06
1000	5	1000000	0.07	0.07	0.27	0.49	0	0.07	0.03
1000	5	1000000	0.07	0.07	0.3	0.49	0	0.07	0

(p) Run 2 Group 8

1000	5	1000000	0.25	0.25	0	0.25	0	0.25	0
1000	5	1000000	0.2425	0.2425	0.03	0.2425	0	0.2425	0
1000	5	1000000	0.235	0.235	0.06	0.235	0	0.235	0
1000	5	1000000	0.2275	0.2275	0.09	0.2275	0	0.2275	0
1000	5	1000000	0.22	0.22	0.12	0.22	0	0.22	0
1000	5	1000000	0.2125	0.2125	0.15	0.2125	0	0.2125	0
1000	5	1000000	0.205	0.205	0.18	0.205	0	0.205	0
1000	5	1000000	0.1975	0.1975	0.21	0.1975	0	0.1975	0
1000	5	1000000	0.19	0.19	0.24	0.19	0	0.19	0
1000	5	1000000	0.1825	0.1825	0.27	0.1825	0	0.1825	0
1000	5	1000000	0.175	0.175	0.3	0.175	0	0.175	0

(q) Run 1 Group 9

1000	5	1000000	0.175	0.175	0	0.175	0	0.175	0.3
1000	5	1000000	0.175	0.175	0.03	0.175	0	0.175	0.27
1000	5	1000000	0.175	0.175	0.06	0.175	0	0.175	0.24
1000	5	1000000	0.175	0.175	0.09	0.175	0	0.175	0.21
1000	5	1000000	0.175	0.175	0.12	0.175	0	0.175	0.18
1000	5	1000000	0.175	0.175	0.15	0.175	0	0.175	0.15
1000	5	1000000	0.175	0.175	0.18	0.175	0	0.175	0.12
1000	5	1000000	0.175	0.175	0.21	0.175	0	0.175	0.09
1000	5	1000000	0.175	0.175	0.24	0.175	0	0.175	0.06
1000	5	1000000	0.175	0.175	0.27	0.175	0	0.175	0.03
1000	5	1000000	0.175	0.175	0.3	0.175	0	0.175	0

(r) Run 2 Group 9

1000	5	1000000	0.4	0.4	0	0.1	0	0.1	0
1000	5	1000000	0.388	0.388	0.03	0.097	0	0.097	0
1000	5	1000000	0.376	0.376	0.06	0.094	0	0.094	0
1000	5	1000000	0.364	0.364	0.09	0.091	0	0.091	0
1000	5	1000000	0.352	0.352	0.12	0.088	0	0.088	0
1000	5	1000000	0.34	0.34	0.15	0.085	0	0.085	0
1000	5	1000000	0.328	0.328	0.18	0.082	0	0.082	0
1000	5	1000000	0.316	0.316	0.21	0.079	0	0.079	0
1000	5	1000000	0.304	0.304	0.24	0.076	0	0.076	0
1000	5	1000000	0.292	0.292	0.27	0.073	0	0.073	0
1000	5	1000000	0.28	0.28	0.3	0.07	0	0.07	0

(s) Run 1 Group 10

1000	5	1000000	0.28	0.28	0	0.07	0	0.07	0.3
1000	5	1000000	0.28	0.28	0.03	0.07	0	0.07	0.27
1000	5	1000000	0.28	0.28	0.06	0.07	0	0.07	0.24
1000	5	1000000	0.28	0.28	0.09	0.07	0	0.07	0.21
1000	5	1000000	0.28	0.28	0.12	0.07	0	0.07	0.18
1000	5	1000000	0.28	0.28	0.15	0.07	0	0.07	0.15
1000	5	1000000	0.28	0.28	0.18	0.07	0	0.07	0.12
1000	5	1000000	0.28	0.28	0.21	0.07	0	0.07	0.09
1000	5	1000000	0.28	0.28	0.24	0.07	0	0.07	0.06
1000	5	1000000	0.28	0.28	0.27	0.07	0	0.07	0.03
1000	5	1000000	0.28	0.28	0.3	0.07	0	0.07	0

(t) Run 2 Group 10

1000	5	1000000	0	0	0.1	0	0	0	0.9
1000	5	1000000	0	0	0.2	0	0	0	0.8
1000	5	1000000	0	0	0.3	0	0	0	0.7
1000	5	1000000	0	0	0.4	0	0	0	0.6
1000	5	1000000	0	0	0.5	0	0	0	0.5
1000	5	1000000	0	0	0.6	0	0	0	0.4
1000	5	1000000	0	0	0.7	0	0	0	0.3
1000	5	1000000	0	0	0.8	0	0	0	0.2
1000	5	1000000	0	0	0.9	0	0	0	0.1
1000	5	1000000	0	0	1	0	0	0	0

(u) Null Experiment

Figure B.3: Add's Experimental Settings

B.4 Delete

1000	5	1000000	0.2	0.2	0.3	0	0	0.3	0
1000	5	1000000	0.194	0.194	0.291	0.03	0	0.291	0
1000	5	1000000	0.188	0.188	0.282	0.06	0	0.282	0
1000	5	1000000	0.182	0.182	0.273	0.09	0	0.273	0
1000	5	1000000	0.176	0.176	0.264	0.12	0	0.264	0
1000	5	1000000	0.17	0.17	0.255	0.15	0	0.255	0
1000	5	1000000	0.164	0.164	0.246	0.18	0	0.246	0
1000	5	1000000	0.158	0.158	0.237	0.21	0	0.237	0
1000	5	1000000	0.152	0.152	0.228	0.24	0	0.228	0
1000	5	1000000	0.146	0.146	0.219	0.27	0	0.219	0
1000	5	1000000	0.14	0.14	0.21	0.3	0	0.21	0
1000	5	1000000	0.45	0.25	0	0	0	0.3	0

(a) Run 1 Group 1

1000	5	1000000	0.14	0.14	0.21	0	0	0.21	0.3
1000	5	1000000	0.14	0.14	0.21	0.03	0	0.21	0.27
1000	5	1000000	0.14	0.14	0.21	0.06	0	0.21	0.24
1000	5	1000000	0.14	0.14	0.21	0.09	0	0.21	0.21
1000	5	1000000	0.14	0.14	0.21	0.12	0	0.21	0.18
1000	5	1000000	0.14	0.14	0.21	0.15	0	0.21	0.15
1000	5	1000000	0.14	0.14	0.21	0.18	0	0.21	0.12
1000	5	1000000	0.14	0.14	0.21	0.21	0	0.21	0.09
1000	5	1000000	0.14	0.14	0.21	0.24	0	0.21	0.06
1000	5	1000000	0.14	0.14	0.21	0.27	0	0.21	0.03
1000	5	1000000	0.14	0.14	0.21	0.3	0	0.21	0

(b) Run 2 Group 1

1000	5	1000000	0.45	0.25	0	0	0	0.3	0
1000	5	1000000	0.4365	0.2425	0	0.03	0	0.291	0
1000	5	1000000	0.423	0.235	0	0.06	0	0.282	0
1000	5	1000000	0.4095	0.2275	0	0.09	0	0.273	0
1000	5	1000000	0.396	0.22	0	0.12	0	0.264	0
1000	5	1000000	0.3825	0.2125	0	0.15	0	0.255	0
1000	5	1000000	0.369	0.205	0	0.18	0	0.246	0
1000	5	1000000	0.3555	0.1975	0	0.21	0	0.237	0
1000	5	1000000	0.342	0.19	0	0.24	0	0.228	0
1000	5	1000000	0.3285	0.1825	0	0.27	0	0.219	0
1000	5	1000000	0.315	0.175	0	0.3	0	0.21	0

(c) Run 1 Group 2

1000	5	1000000	0.315	0.175	0	0	0	0.21	0.3
1000	5	1000000	0.315	0.175	0	0.03	0	0.21	0.27
1000	5	1000000	0.315	0.175	0	0.06	0	0.21	0.24
1000	5	1000000	0.315	0.175	0	0.09	0	0.21	0.21
1000	5	1000000	0.315	0.175	0	0.12	0	0.21	0.18
1000	5	1000000	0.315	0.175	0	0.15	0	0.21	0.15
1000	5	1000000	0.315	0.175	0	0.18	0	0.21	0.12
1000	5	1000000	0.315	0.175	0	0.21	0	0.21	0.09
1000	5	1000000	0.315	0.175	0	0.24	0	0.21	0.06
1000	5	1000000	0.315	0.175	0	0.27	0	0.21	0.03
1000	5	1000000	0.315	0.175	0	0.3	0	0.21	0

(d) Run 2 Group 2

1000	5	1000000	0.2	0.2	0.4	0	0	0.2	0
1000	5	1000000	0.194	0.194	0.388	0.03	0	0.194	0
1000	5	1000000	0.188	0.188	0.376	0.06	0	0.188	0
1000	5	1000000	0.182	0.182	0.364	0.09	0	0.182	0
1000	5	1000000	0.176	0.176	0.352	0.12	0	0.176	0
1000	5	1000000	0.17	0.17	0.34	0.15	0	0.17	0
1000	5	1000000	0.164	0.164	0.328	0.18	0	0.164	0
1000	5	1000000	0.158	0.158	0.316	0.21	0	0.158	0
1000	5	1000000	0.152	0.152	0.304	0.24	0	0.152	0
1000	5	1000000	0.146	0.146	0.292	0.27	0	0.146	0
1000	5	1000000	0.14	0.14	0.28	0.3	0	0.14	0

(e) Run 1 Group 3

1000	5	1000000	0.14	0.14	0.28	0	0	0.14	0.3
1000	5	1000000	0.14	0.14	0.28	0.03	0	0.14	0.27
1000	5	1000000	0.14	0.14	0.28	0.06	0	0.14	0.24
1000	5	1000000	0.14	0.14	0.28	0.09	0	0.14	0.21
1000	5	1000000	0.14	0.14	0.28	0.12	0	0.14	0.18
1000	5	1000000	0.14	0.14	0.28	0.15	0	0.14	0.15
1000	5	1000000	0.14	0.14	0.28	0.18	0	0.14	0.12
1000	5	1000000	0.14	0.14	0.28	0.21	0	0.14	0.09
1000	5	1000000	0.14	0.14	0.28	0.24	0	0.14	0.06
1000	5	1000000	0.14	0.14	0.28	0.27	0	0.14	0.03
1000	5	1000000	0.14	0.14	0.28	0.3	0	0.14	0

(f) Run 2 Group 3

1000	5	1000000	0.3	0.3	0.1	0	0	0.3	0
1000	5	1000000	0.291	0.291	0.097	0.03	0	0.291	0
1000	5	1000000	0.282	0.282	0.094	0.06	0	0.282	0
1000	5	1000000	0.273	0.273	0.091	0.09	0	0.273	0
1000	5	1000000	0.264	0.264	0.088	0.12	0	0.264	0
1000	5	1000000	0.255	0.255	0.085	0.15	0	0.255	0
1000	5	1000000	0.246	0.246	0.082	0.18	0	0.246	0
1000	5	1000000	0.237	0.237	0.079	0.21	0	0.237	0
1000	5	1000000	0.228	0.228	0.076	0.24	0	0.228	0
1000	5	1000000	0.219	0.219	0.073	0.27	0	0.219	0
1000	5	1000000	0.21	0.21	0.07	0.3	0	0.21	0
1000	5	1000000	0.4	0.3	0.15	0	0	0.15	0

(g) Run 1 Group 4

1000	5	1000000	0.21	0.21	0.07	0	0	0.21	0.3
1000	5	1000000	0.21	0.21	0.07	0.03	0	0.21	0.27
1000	5	1000000	0.21	0.21	0.07	0.06	0	0.21	0.24
1000	5	1000000	0.21	0.21	0.07	0.09	0	0.21	0.21
1000	5	1000000	0.21	0.21	0.07	0.12	0	0.21	0.18
1000	5	1000000	0.21	0.21	0.07	0.15	0	0.21	0.15
1000	5	1000000	0.21	0.21	0.07	0.18	0	0.21	0.12
1000	5	1000000	0.21	0.21	0.07	0.21	0	0.21	0.09
1000	5	1000000	0.21	0.21	0.07	0.24	0	0.21	0.06
1000	5	1000000	0.21	0.21	0.07	0.27	0	0.21	0.03
1000	5	1000000	0.21	0.21	0.07	0.3	0	0.21	0

(h) Run 2 Group 4

1000	5	1000000	0.4	0.3	0.15	0	0	0.15	0
1000	5	1000000	0.388	0.291	0.1455	0.03	0	0.1455	0
1000	5	1000000	0.376	0.282	0.141	0.06	0	0.141	0
1000	5	1000000	0.364	0.273	0.1365	0.09	0	0.1365	0
1000	5	1000000	0.352	0.264	0.132	0.12	0	0.132	0
1000	5	1000000	0.34	0.255	0.1275	0.15	0	0.1275	0
1000	5	1000000	0.328	0.246	0.123	0.18	0	0.123	0
1000	5	1000000	0.316	0.237	0.1185	0.21	0	0.1185	0
1000	5	1000000	0.304	0.228	0.114	0.24	0	0.114	0
1000	5	1000000	0.292	0.219	0.1095	0.27	0	0.1095	0
1000	5	1000000	0.28	0.21	0.105	0.3	0	0.105	0

(i) Run 1 Group 5

1000	5	1000000	0.28	0.21	0.105	0	0	0.105	0.3
1000	5	1000000	0.28	0.21	0.105	0.03	0	0.105	0.27
1000	5	1000000	0.28	0.21	0.105	0.06	0	0.105	0.24
1000	5	1000000	0.28	0.21	0.105	0.09	0	0.105	0.21
1000	5	1000000	0.28	0.21	0.105	0.12	0	0.105	0.18
1000	5	1000000	0.28	0.21	0.105	0.15	0	0.105	0.15
1000	5	1000000	0.28	0.21	0.105	0.18	0	0.105	0.12
1000	5	1000000	0.28	0.21	0.105	0.21	0	0.105	0.09
1000	5	1000000	0.28	0.21	0.105	0.24	0	0.105	0.06
1000	5	1000000	0.28	0.21	0.105	0.27	0	0.105	0.03
1000	5	1000000	0.28	0.21	0.105	0.3	0	0.105	0

(j) Run 2 Group 5

1000	5	1000000	0.2	0.6	0.1	0	0	0.1	0
1000	5	1000000	0.194	0.582	0.097	0.03	0	0.097	0
1000	5	1000000	0.188	0.564	0.094	0.06	0	0.094	0
1000	5	1000000	0.182	0.546	0.091	0.09	0	0.091	0
1000	5	1000000	0.176	0.528	0.088	0.12	0	0.088	0
1000	5	1000000	0.17	0.51	0.085	0.15	0	0.085	0
1000	5	1000000	0.164	0.492	0.082	0.18	0	0.082	0
1000	5	1000000	0.158	0.474	0.079	0.21	0	0.079	0
1000	5	1000000	0.152	0.456	0.076	0.24	0	0.076	0
1000	5	1000000	0.146	0.438	0.073	0.27	0	0.073	0
1000	5	1000000	0.14	0.42	0.07	0.3	0	0.07	0

(k) Run 1 Group 6

1000	5	1000000	0.14	0.42	0.07	0	0	0.07	0.3
1000	5	1000000	0.14	0.42	0.07	0.03	0	0.07	0.27
1000	5	1000000	0.14	0.42	0.07	0.06	0	0.07	0.24
1000	5	1000000	0.14	0.42	0.07	0.09	0	0.07	0.21
1000	5	1000000	0.14	0.42	0.07	0.12	0	0.07	0.18
1000	5	1000000	0.14	0.42	0.07	0.15	0	0.07	0.15
1000	5	1000000	0.14	0.42	0.07	0.18	0	0.07	0.12
1000	5	1000000	0.14	0.42	0.07	0.21	0	0.07	0.09
1000	5	1000000	0.14	0.42	0.07	0.24	0	0.07	0.06
1000	5	1000000	0.14	0.42	0.07	0.27	0	0.07	0.03
1000	5	1000000	0.14	0.42	0.07	0.3	0	0.07	0

(l) Run 2 Group 6

1000	5	1000000	0.2	0.4	0.2	0	0	0.2	0
1000	5	1000000	0.194	0.388	0.194	0.03	0	0.194	0
1000	5	1000000	0.188	0.376	0.188	0.06	0	0.188	0
1000	5	1000000	0.182	0.364	0.182	0.09	0	0.182	0
1000	5	1000000	0.176	0.352	0.176	0.12	0	0.176	0
1000	5	1000000	0.17	0.34	0.17	0.15	0	0.17	0
1000	5	1000000	0.164	0.328	0.164	0.18	0	0.164	0
1000	5	1000000	0.158	0.316	0.158	0.21	0	0.158	0
1000	5	1000000	0.152	0.304	0.152	0.24	0	0.152	0
1000	5	1000000	0.146	0.292	0.146	0.27	0	0.146	0
1000	5	1000000	0.14	0.28	0.14	0.3	0	0.14	0

(m) Run 1 Group 7

1000	5	1000000	0.14	0.28	0.14	0	0	0.14	0.3
1000	5	1000000	0.14	0.28	0.14	0.03	0	0.14	0.27
1000	5	1000000	0.14	0.28	0.14	0.06	0	0.14	0.24
1000	5	1000000	0.14	0.28	0.14	0.09	0	0.14	0.21
1000	5	1000000	0.14	0.28	0.14	0.12	0	0.14	0.18
1000	5	1000000	0.14	0.28	0.14	0.15	0	0.14	0.15
1000	5	1000000	0.14	0.28	0.14	0.18	0	0.14	0.12
1000	5	1000000	0.14	0.28	0.14	0.21	0	0.14	0.09
1000	5	1000000	0.14	0.28	0.14	0.24	0	0.14	0.06
1000	5	1000000	0.14	0.28	0.14	0.27	0	0.14	0.03
1000	5	1000000	0.14	0.28	0.14	0.3	0	0.14	0

(n) Run 2 Group 7

1000	5	1000000	0.1	0.1	0.7	0	0	0.1	0
1000	5	1000000	0.097	0.097	0.679	0.03	0	0.097	0
1000	5	1000000	0.094	0.094	0.658	0.06	0	0.094	0
1000	5	1000000	0.091	0.091	0.637	0.09	0	0.091	0
1000	5	1000000	0.088	0.088	0.616	0.12	0	0.088	0
1000	5	1000000	0.085	0.085	0.595	0.15	0	0.085	0
1000	5	1000000	0.082	0.082	0.574	0.18	0	0.082	0
1000	5	1000000	0.079	0.079	0.553	0.21	0	0.079	0
1000	5	1000000	0.076	0.076	0.532	0.24	0	0.076	0
1000	5	1000000	0.073	0.073	0.511	0.27	0	0.073	0
1000	5	1000000	0.07	0.07	0.49	0.3	0	0.07	0

(o) Run 1 Group 8

1000	5	1000000	0.07	0.07	0.49	0	0	0.07	0.3
1000	5	1000000	0.07	0.07	0.49	0.03	0	0.07	0.27
1000	5	1000000	0.07	0.07	0.49	0.06	0	0.07	0.24
1000	5	1000000	0.07	0.07	0.49	0.09	0	0.07	0.21
1000	5	1000000	0.07	0.07	0.49	0.12	0	0.07	0.18
1000	5	1000000	0.07	0.07	0.49	0.15	0	0.07	0.15
1000	5	1000000	0.07	0.07	0.49	0.18	0	0.07	0.12
1000	5	1000000	0.07	0.07	0.49	0.21	0	0.07	0.09
1000	5	1000000	0.07	0.07	0.49	0.24	0	0.07	0.06
1000	5	1000000	0.07	0.07	0.49	0.27	0	0.07	0.03
1000	5	1000000	0.07	0.07	0.49	0.3	0	0.07	0

(p) Run 2 Group 8

1000	5	1000000	0.25	0.25	0.25	0	0	0.25	0
1000	5	1000000	0.2425	0.2425	0.2425	0.03	0	0.2425	0
1000	5	1000000	0.235	0.235	0.235	0.06	0	0.235	0
1000	5	1000000	0.2275	0.2275	0.2275	0.09	0	0.2275	0
1000	5	1000000	0.22	0.22	0.22	0.12	0	0.22	0
1000	5	1000000	0.2125	0.2125	0.2125	0.15	0	0.2125	0
1000	5	1000000	0.205	0.205	0.205	0.18	0	0.205	0
1000	5	1000000	0.1975	0.1975	0.1975	0.21	0	0.1975	0
1000	5	1000000	0.19	0.19	0.19	0.24	0	0.19	0
1000	5	1000000	0.1825	0.1825	0.1825	0.27	0	0.1825	0
1000	5	1000000	0.175	0.175	0.175	0.3	0	0.175	0

(q) Run 1 Group 9

1000	5	1000000	0.175	0.175	0.175	0	0	0.175	0.3
1000	5	1000000	0.175	0.175	0.175	0.03	0	0.175	0.27
1000	5	1000000	0.175	0.175	0.175	0.06	0	0.175	0.24
1000	5	1000000	0.175	0.175	0.175	0.09	0	0.175	0.21
1000	5	1000000	0.175	0.175	0.175	0.12	0	0.175	0.18
1000	5	1000000	0.175	0.175	0.175	0.15	0	0.175	0.15
1000	5	1000000	0.175	0.175	0.175	0.18	0	0.175	0.12
1000	5	1000000	0.175	0.175	0.175	0.21	0	0.175	0.09
1000	5	1000000	0.175	0.175	0.175	0.24	0	0.175	0.06
1000	5	1000000	0.175	0.175	0.175	0.27	0	0.175	0.03
1000	5	1000000	0.175	0.175	0.175	0.3	0	0.175	0

(r) Run 2 Group 9

1000	5	1000000	0.4	0.4	0.1	0	0	0.1	0
1000	5	1000000	0.388	0.388	0.097	0.03	0	0.097	0
1000	5	1000000	0.376	0.376	0.094	0.06	0	0.094	0
1000	5	1000000	0.364	0.364	0.091	0.09	0	0.091	0
1000	5	1000000	0.352	0.352	0.088	0.12	0	0.088	0
1000	5	1000000	0.34	0.34	0.085	0.15	0	0.085	0
1000	5	1000000	0.328	0.328	0.082	0.18	0	0.082	0
1000	5	1000000	0.316	0.316	0.079	0.21	0	0.079	0
1000	5	1000000	0.304	0.304	0.076	0.24	0	0.076	0
1000	5	1000000	0.292	0.292	0.073	0.27	0	0.073	0
1000	5	1000000	0.28	0.28	0.07	0.3	0	0.07	0

(s) Run 1 Group 10

1000	5	1000000	0.28	0.28	0.07	0	0	0.07	0.3
1000	5	1000000	0.28	0.28	0.07	0.03	0	0.07	0.27
1000	5	1000000	0.28	0.28	0.07	0.06	0	0.07	0.24
1000	5	1000000	0.28	0.28	0.07	0.09	0	0.07	0.21
1000	5	1000000	0.28	0.28	0.07	0.12	0	0.07	0.18
1000	5	1000000	0.28	0.28	0.07	0.15	0	0.07	0.15
1000	5	1000000	0.28	0.28	0.07	0.18	0	0.07	0.12
1000	5	1000000	0.28	0.28	0.07	0.21	0	0.07	0.09
1000	5	1000000	0.28	0.28	0.07	0.24	0	0.07	0.06
1000	5	1000000	0.28	0.28	0.07	0.27	0	0.07	0.03
1000	5	1000000	0.28	0.28	0.07	0.3	0	0.07	0

(t) Run 2 Group 10

1000	5	1000000	0	0	0	0.1	0	0	0.9
1000	5	1000000	0	0	0	0.2	0	0	0.8
1000	5	1000000	0	0	0	0.3	0	0	0.7
1000	5	1000000	0	0	0	0.4	0	0	0.6
1000	5	1000000	0	0	0	0.5	0	0	0.5
1000	5	1000000	0	0	0	0.6	0	0	0.4
1000	5	1000000	0	0	0	0.7	0	0	0.3
1000	5	1000000	0	0	0	0.8	0	0	0.2
1000	5	1000000	0	0	0	0.9	0	0	0.1
1000	5	1000000	0	0	0	1	0	0	0

(u) Null Experiment

Figure B.4: Delete's Experimental Settings

B.5 Swap

1000	5	1000000	0.2	0.2	0.2	0.2	0	0.2	0
1000	5	1000000	0.194	0.194	0.194	0.194	0.03	0.194	0
1000	5	1000000	0.188	0.188	0.188	0.188	0.06	0.188	0
1000	5	1000000	0.182	0.182	0.182	0.182	0.09	0.182	0
1000	5	1000000	0.176	0.176	0.176	0.176	0.12	0.176	0
1000	5	1000000	0.17	0.17	0.17	0.17	0.15	0.17	0
1000	5	1000000	0.164	0.164	0.164	0.164	0.18	0.164	0
1000	5	1000000	0.158	0.158	0.158	0.158	0.21	0.158	0
1000	5	1000000	0.152	0.152	0.152	0.152	0.24	0.152	0
1000	5	1000000	0.146	0.146	0.146	0.146	0.27	0.146	0
1000	5	1000000	0.14	0.14	0.14	0.14	0.3	0.14	0

(a) Run 1 Group 1

1000	5	1000000	0.14	0.14	0.14	0.14	0	0.14	0.3
1000	5	1000000	0.14	0.14	0.14	0.14	0.03	0.14	0.27
1000	5	1000000	0.14	0.14	0.14	0.14	0.06	0.14	0.24
1000	5	1000000	0.14	0.14	0.14	0.14	0.09	0.14	0.21
1000	5	1000000	0.14	0.14	0.14	0.14	0.12	0.14	0.18
1000	5	1000000	0.14	0.14	0.14	0.14	0.15	0.14	0.15
1000	5	1000000	0.14	0.14	0.14	0.14	0.18	0.14	0.12
1000	5	1000000	0.14	0.14	0.14	0.14	0.21	0.14	0.09
1000	5	1000000	0.14	0.14	0.14	0.14	0.24	0.14	0.06
1000	5	1000000	0.14	0.14	0.14	0.14	0.27	0.14	0.03
1000	5	1000000	0.14	0.14	0.14	0.14	0.3	0.14	0

(b) Run 2 Group 1

1000	5	1000000	0.05	0.1	0.25	0.25	0	0.35	0
1000	5	1000000	0.0485	0.097	0.2425	0.2425	0.03	0.3395	0
1000	5	1000000	0.047	0.094	0.235	0.235	0.06	0.329	0
1000	5	1000000	0.0455	0.091	0.2275	0.2275	0.09	0.3185	0
1000	5	1000000	0.044	0.088	0.22	0.22	0.12	0.308	0
1000	5	1000000	0.0425	0.085	0.2125	0.2125	0.15	0.2975	0
1000	5	1000000	0.041	0.082	0.205	0.205	0.18	0.287	0
1000	5	1000000	0.0395	0.079	0.1975	0.1975	0.21	0.2765	0
1000	5	1000000	0.038	0.076	0.19	0.19	0.24	0.266	0
1000	5	1000000	0.0365	0.073	0.1825	0.1825	0.27	0.2555	0
1000	5	1000000	0.035	0.07	0.175	0.175	0.3	0.245	0

(c) Run 1 Group 2

1000	5	1000000	0.035	0.07	0.175	0.175	0	0.245	0.3
1000	5	1000000	0.035	0.07	0.175	0.175	0.03	0.245	0.27
1000	5	1000000	0.035	0.07	0.175	0.175	0.06	0.245	0.24
1000	5	1000000	0.035	0.07	0.175	0.175	0.09	0.245	0.21
1000	5	1000000	0.035	0.07	0.175	0.175	0.12	0.245	0.18
1000	5	1000000	0.035	0.07	0.175	0.175	0.15	0.245	0.15
1000	5	1000000	0.035	0.07	0.175	0.175	0.18	0.245	0.12
1000	5	1000000	0.035	0.07	0.175	0.175	0.21	0.245	0.09
1000	5	1000000	0.035	0.07	0.175	0.175	0.24	0.245	0.06
1000	5	1000000	0.035	0.07	0.175	0.175	0.27	0.245	0.03
1000	5	1000000	0.035	0.07	0.175	0.175	0.3	0.245	0

(d) Run 2 Group 2

1000	5	1000000	0.45	0.45	0.05	0.05	0	0	0
1000	5	1000000	0.4365	0.4365	0.0485	0.0485	0.03	0	0
1000	5	1000000	0.423	0.423	0.047	0.047	0.06	0	0
1000	5	1000000	0.4095	0.4095	0.0455	0.0455	0.09	0	0
1000	5	1000000	0.396	0.396	0.044	0.044	0.12	0	0
1000	5	1000000	0.3825	0.3825	0.0425	0.0425	0.15	0	0
1000	5	1000000	0.369	0.369	0.041	0.041	0.18	0	0
1000	5	1000000	0.3555	0.3555	0.0395	0.0395	0.21	0	0
1000	5	1000000	0.342	0.342	0.038	0.038	0.24	0	0
1000	5	1000000	0.3285	0.3285	0.0365	0.0365	0.27	0	0
1000	5	1000000	0.315	0.315	0.035	0.035	0.3	0	0

(e) Run 1 Group 3

1000	5	1000000	0.315	0.315	0.035	0.035	0	0	0.3
1000	5	1000000	0.315	0.315	0.035	0.035	0.03	0	0.27
1000	5	1000000	0.315	0.315	0.035	0.035	0.06	0	0.24
1000	5	1000000	0.315	0.315	0.035	0.035	0.09	0	0.21
1000	5	1000000	0.315	0.315	0.035	0.035	0.12	0	0.18
1000	5	1000000	0.315	0.315	0.035	0.035	0.15	0	0.15
1000	5	1000000	0.315	0.315	0.035	0.035	0.18	0	0.12
1000	5	1000000	0.315	0.315	0.035	0.035	0.21	0	0.09
1000	5	1000000	0.315	0.315	0.035	0.035	0.24	0	0.06
1000	5	1000000	0.315	0.315	0.035	0.035	0.27	0	0.03
1000	5	1000000	0.315	0.315	0.035	0.035	0.3	0	0

(f) Run 2 Group 3

1000	5	1000000	0.4	0.1	0.15	0.15	0	0.2	0
1000	5	1000000	0.388	0.097	0.1455	0.1455	0.03	0.194	0
1000	5	1000000	0.376	0.094	0.141	0.141	0.06	0.188	0
1000	5	1000000	0.364	0.091	0.1365	0.1365	0.09	0.182	0
1000	5	1000000	0.352	0.088	0.132	0.132	0.12	0.176	0
1000	5	1000000	0.34	0.085	0.1275	0.1275	0.15	0.17	0
1000	5	1000000	0.328	0.082	0.123	0.123	0.18	0.164	0
1000	5	1000000	0.316	0.079	0.1185	0.1185	0.21	0.158	0
1000	5	1000000	0.304	0.076	0.114	0.114	0.24	0.152	0
1000	5	1000000	0.292	0.073	0.1095	0.1095	0.27	0.146	0
1000	5	1000000	0.28	0.07	0.105	0.105	0.3	0.14	0

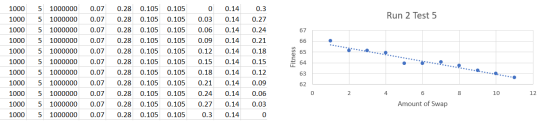
(g) Run 1 Group 4

1000	5	1000000	0.28	0.07	0.105	0.105	0	0.14	0.3
1000	5	1000000	0.28	0.07	0.105	0.105	0.03	0.14	0.27
1000	5	1000000	0.28	0.07	0.105	0.105	0.06	0.14	0.24
1000	5	1000000	0.28	0.07	0.105	0.105	0.09	0.14	0.21
1000	5	1000000	0.28	0.07	0.105	0.105	0.12	0.14	0.18
1000	5	1000000	0.28	0.07	0.105	0.105	0.15	0.14	0.15
1000	5	1000000	0.28	0.07	0.105	0.105	0.18	0.14	0.12
1000	5	1000000	0.28	0.07	0.105	0.105	0.21	0.14	0.09
1000	5	1000000	0.28	0.07	0.105	0.105	0.24	0.14	0.06
1000	5	1000000	0.28	0.07	0.105	0.105	0.27	0.14	0.03
1000	5	1000000	0.28	0.07	0.105	0.105	0.3	0.14	0

(h) Run 2 Group 4

1000	5	1000000	0.1	0.4	0.15	0.15	0	0.2	0
1000	5	1000000	0.097	0.388	0.1455	0.1455	0.03	0.194	0
1000	5	1000000	0.094	0.376	0.141	0.141	0.06	0.188	0
1000	5	1000000	0.091	0.364	0.1365	0.1365	0.09	0.182	0
1000	5	1000000	0.088	0.352	0.132	0.132	0.12	0.176	0
1000	5	1000000	0.085	0.34	0.1275	0.1275	0.15	0.17	0
1000	5	1000000	0.082	0.328	0.123	0.123	0.18	0.164	0
1000	5	1000000	0.079	0.316	0.1185	0.1185	0.21	0.158	0
1000	5	1000000	0.076	0.304	0.114	0.114	0.24	0.152	0
1000	5	1000000	0.073	0.292	0.1095	0.1095	0.27	0.146	0
1000	5	1000000	0.07	0.28	0.105	0.105	0.3	0.14	0

(i) Run 1 Group 5



(j) Run 2 Group 5

1000	5	1000000	0.2	0.1	0.2	0.2	0	0.3	0
1000	5	1000000	0.194	0.097	0.194	0.194	0.03	0.291	0
1000	5	1000000	0.188	0.094	0.188	0.188	0.06	0.282	0
1000	5	1000000	0.182	0.091	0.182	0.182	0.09	0.273	0
1000	5	1000000	0.176	0.088	0.176	0.176	0.12	0.264	0
1000	5	1000000	0.17	0.085	0.17	0.17	0.15	0.255	0
1000	5	1000000	0.164	0.082	0.164	0.164	0.18	0.246	0
1000	5	1000000	0.158	0.079	0.158	0.158	0.21	0.237	0
1000	5	1000000	0.152	0.076	0.152	0.152	0.24	0.228	0
1000	5	1000000	0.146	0.073	0.146	0.146	0.27	0.219	0
1000	5	1000000	0.14	0.07	0.14	0.14	0.3	0.21	0

(k) Run 1 Group 6

1000	5	1000000	0.14	0.07	0.14	0.14	0	0.21	0.3
1000	5	1000000	0.14	0.07	0.14	0.14	0.03	0.21	0.27
1000	5	1000000	0.14	0.07	0.14	0.14	0.06	0.21	0.24
1000	5	1000000	0.14	0.07	0.14	0.14	0.09	0.21	0.21
1000	5	1000000	0.14	0.07	0.14	0.14	0.12	0.21	0.18
1000	5	1000000	0.14	0.07	0.14	0.14	0.15	0.21	0.15
1000	5	1000000	0.14	0.07	0.14	0.14	0.18	0.21	0.12
1000	5	1000000	0.14	0.07	0.14	0.14	0.21	0.21	0.09
1000	5	1000000	0.14	0.07	0.14	0.14	0.24	0.21	0.06
1000	5	1000000	0.14	0.07	0.14	0.14	0.27	0.21	0.03
1000	5	1000000	0.14	0.07	0.14	0.14	0.3	0.21	0

(l) Run 2 Group 6

1000	5	1000000	0.3	0.1	0.15	0.15	0	0.3	0
1000	5	1000000	0.291	0.097	0.1455	0.1455	0.03	0.291	0
1000	5	1000000	0.282	0.094	0.141	0.141	0.06	0.282	0
1000	5	1000000	0.273	0.091	0.1365	0.1365	0.09	0.273	0
1000	5	1000000	0.264	0.088	0.132	0.132	0.12	0.264	0
1000	5	1000000	0.255	0.085	0.1275	0.1275	0.15	0.255	0
1000	5	1000000	0.246	0.082	0.123	0.123	0.18	0.246	0
1000	5	1000000	0.237	0.079	0.1185	0.1185	0.21	0.237	0
1000	5	1000000	0.228	0.076	0.114	0.114	0.24	0.228	0
1000	5	1000000	0.219	0.073	0.1095	0.1095	0.27	0.219	0
1000	5	1000000	0.21	0.07	0.105	0.105	0.3	0.21	0

(m) Run 1 Group 7

1000	5	1000000	0.21	0.07	0.105	0.105	0	0.21	0.3
1000	5	1000000	0.21	0.07	0.105	0.105	0.03	0.21	0.27
1000	5	1000000	0.21	0.07	0.105	0.105	0.06	0.21	0.24
1000	5	1000000	0.21	0.07	0.105	0.105	0.09	0.21	0.21
1000	5	1000000	0.21	0.07	0.105	0.105	0.12	0.21	0.18
1000	5	1000000	0.21	0.07	0.105	0.105	0.15	0.21	0.15
1000	5	1000000	0.21	0.07	0.105	0.105	0.18	0.21	0.12
1000	5	1000000	0.21	0.07	0.105	0.105	0.21	0.21	0.09
1000	5	1000000	0.21	0.07	0.105	0.105	0.24	0.21	0.06
1000	5	1000000	0.21	0.07	0.105	0.105	0.27	0.21	0.03
1000	5	1000000	0.21	0.07	0.105	0.105	0.3	0.21	0

(n) Run 2 Group 7

1000	5	1000000	0.5	0.2	0.15	0.15	0	0	0
1000	5	1000000	0.485	0.194	0.1455	0.1455	0.03	0	0
1000	5	1000000	0.47	0.188	0.141	0.141	0.06	0	0
1000	5	1000000	0.455	0.182	0.1365	0.1365	0.09	0	0
1000	5	1000000	0.44	0.176	0.132	0.132	0.12	0	0
1000	5	1000000	0.425	0.17	0.1275	0.1275	0.15	0	0
1000	5	1000000	0.41	0.164	0.123	0.123	0.18	0	0
1000	5	1000000	0.395	0.158	0.1185	0.1185	0.21	0	0
1000	5	1000000	0.38	0.152	0.114	0.114	0.24	0	0
1000	5	1000000	0.365	0.146	0.1095	0.1095	0.27	0	0
1000	5	1000000	0.35	0.14	0.105	0.105	0.3	0	0

(o) Run 1 Group 8

1000	5	1000000	0.35	0.14	0.105	0.105	0	0	0.3
1000	5	1000000	0.35	0.14	0.105	0.105	0.03	0	0.27
1000	5	1000000	0.35	0.14	0.105	0.105	0.06	0	0.24
1000	5	1000000	0.35	0.14	0.105	0.105	0.09	0	0.21
1000	5	1000000	0.35	0.14	0.105	0.105	0.12	0	0.18
1000	5	1000000	0.35	0.14	0.105	0.105	0.15	0	0.15
1000	5	1000000	0.35	0.14	0.105	0.105	0.18	0	0.12
1000	5	1000000	0.35	0.14	0.105	0.105	0.21	0	0.09
1000	5	1000000	0.35	0.14	0.105	0.105	0.24	0	0.06
1000	5	1000000	0.35	0.14	0.105	0.105	0.27	0	0.03
1000	5	1000000	0.35	0.14	0.105	0.105	0.3	0	0

(p) Run 2 Group 8

1000	5	1000000	0.25	0.25	0.15	0.15	0	0.2	0
1000	5	1000000	0.2425	0.2425	0.1455	0.1455	0.03	0.194	0
1000	5	1000000	0.235	0.235	0.141	0.141	0.06	0.188	0
1000	5	1000000	0.2275	0.2275	0.1365	0.1365	0.09	0.182	0
1000	5	1000000	0.22	0.22	0.132	0.132	0.12	0.176	0
1000	5	1000000	0.2125	0.2125	0.1275	0.1275	0.15	0.17	0
1000	5	1000000	0.205	0.205	0.123	0.123	0.18	0.164	0
1000	5	1000000	0.1975	0.1975	0.1185	0.1185	0.21	0.158	0
1000	5	1000000	0.19	0.19	0.114	0.114	0.24	0.152	0
1000	5	1000000	0.1825	0.1825	0.1095	0.1095	0.27	0.146	0
1000	5	1000000	0.175	0.175	0.105	0.105	0.3	0.14	0

(q) Run 1 Group 9

1000	5	1000000	0.175	0.175	0.105	0.105	0	0.14	0.3
1000	5	1000000	0.175	0.175	0.105	0.105	0.03	0.14	0.27
1000	5	1000000	0.175	0.175	0.105	0.105	0.06	0.14	0.24
1000	5	1000000	0.175	0.175	0.105	0.105	0.09	0.14	0.21
1000	5	1000000	0.175	0.175	0.105	0.105	0.12	0.14	0.18
1000	5	1000000	0.175	0.175	0.105	0.105	0.15	0.14	0.15
1000	5	1000000	0.175	0.175	0.105	0.105	0.18	0.14	0.12
1000	5	1000000	0.175	0.175	0.105	0.105	0.21	0.14	0.09
1000	5	1000000	0.175	0.175	0.105	0.105	0.24	0.14	0.06
1000	5	1000000	0.175	0.175	0.105	0.105	0.27	0.14	0.03
1000	5	1000000	0.175	0.175	0.105	0.105	0.3	0.14	0

(r) Run 2 Group 9

1000	5	1000000	0.1	0.1	0.15	0.15	0	0.5	0
1000	5	1000000	0.097	0.097	0.1455	0.1455	0.03	0.485	0
1000	5	1000000	0.094	0.094	0.141	0.141	0.06	0.47	0
1000	5	1000000	0.091	0.091	0.1365	0.1365	0.09	0.455	0
1000	5	1000000	0.088	0.088	0.132	0.132	0.12	0.44	0
1000	5	1000000	0.085	0.085	0.1275	0.1275	0.15	0.425	0
1000	5	1000000	0.082	0.082	0.123	0.123	0.18	0.41	0
1000	5	1000000	0.079	0.079	0.1185	0.1185	0.21	0.395	0
1000	5	1000000	0.076	0.076	0.114	0.114	0.24	0.38	0
1000	5	1000000	0.073	0.073	0.1095	0.1095	0.27	0.365	0
1000	5	1000000	0.07	0.07	0.105	0.105	0.3	0.35	0

(s) Run 1 Group 10

1000	5	1000000	0.07	0.07	0.105	0.105	0	0.35	0.3
1000	5	1000000	0.07	0.07	0.105	0.105	0.03	0.35	0.27
1000	5	1000000	0.07	0.07	0.105	0.105	0.06	0.35	0.24
1000	5	1000000	0.07	0.07	0.105	0.105	0.09	0.35	0.21
1000	5	1000000	0.07	0.07	0.105	0.105	0.12	0.35	0.18
1000	5	1000000	0.07	0.07	0.105	0.105	0.15	0.35	0.15
1000	5	1000000	0.07	0.07	0.105	0.105	0.18	0.35	0.12
1000	5	1000000	0.07	0.07	0.105	0.105	0.21	0.35	0.09
1000	5	1000000	0.07	0.07	0.105	0.105	0.24	0.35	0.06
1000	5	1000000	0.07	0.07	0.105	0.105	0.27	0.35	0.03
1000	5	1000000	0.07	0.07	0.105	0.105	0.3	0.35	0

(t) Run 2 Group 10

1000	5	1000000	0	0	0	0	0.1	0	0.9
1000	5	1000000	0	0	0	0	0.2	0	0.8
1000	5	1000000	0	0	0	0	0.3	0	0.7
1000	5	1000000	0	0	0	0	0.4	0	0.6
1000	5	1000000	0	0	0	0	0.5	0	0.5
1000	5	1000000	0	0	0	0	0.6	0	0.4
1000	5	1000000	0	0	0	0	0.7	0	0.3
1000	5	1000000	0	0	0	0	0.8	0	0.2
1000	5	1000000	0	0	0	0	0.9	0	0.1
1000	5	1000000	0	0	0	0	1	0	0

(u) Null Experiment

Figure B.5: Swap's Experimental Settings

B.6 Local Toggle

1000	5	1000000	0.3	0.2	0.2	0.3	0	0	0
1000	5	1000000	0.291	0.194	0.194	0.291	0	0.03	0
1000	5	1000000	0.282	0.188	0.188	0.282	0	0.06	0
1000	5	1000000	0.273	0.182	0.182	0.273	0	0.09	0
1000	5	1000000	0.264	0.176	0.176	0.264	0	0.12	0
1000	5	1000000	0.255	0.17	0.17	0.255	0	0.15	0
1000	5	1000000	0.246	0.164	0.164	0.246	0	0.18	0
1000	5	1000000	0.237	0.158	0.158	0.237	0	0.21	0
1000	5	1000000	0.228	0.152	0.152	0.228	0	0.24	0
1000	5	1000000	0.219	0.146	0.146	0.219	0	0.27	0
1000	5	1000000	0.21	0.14	0.14	0.21	0	0.3	0

(a) Run 1 Group 1

1000	5	1000000	0.21	0.14	0.14	0.21	0	0	0.3
1000	5	1000000	0.21	0.14	0.14	0.21	0	0.03	0.27
1000	5	1000000	0.21	0.14	0.14	0.21	0	0.06	0.24
1000	5	1000000	0.21	0.14	0.14	0.21	0	0.09	0.21
1000	5	1000000	0.21	0.14	0.14	0.21	0	0.12	0.18
1000	5	1000000	0.21	0.14	0.14	0.21	0	0.15	0.15
1000	5	1000000	0.21	0.14	0.14	0.21	0	0.18	0.12
1000	5	1000000	0.21	0.14	0.14	0.21	0	0.21	0.09
1000	5	1000000	0.21	0.14	0.14	0.21	0	0.24	0.06
1000	5	1000000	0.21	0.14	0.14	0.21	0	0.27	0.03
1000	5	1000000	0.21	0.14	0.14	0.21	0	0.3	0

(b) Run 2 Group 1

1000	5	1000000	0	0.45	0.25	0	0.3	0	0
1000	5	1000000	0	0.4365	0.2425	0	0.291	0.03	0
1000	5	1000000	0	0.423	0.235	0	0.282	0.06	0
1000	5	1000000	0	0.4095	0.2275	0	0.273	0.09	0
1000	5	1000000	0	0.396	0.22	0	0.264	0.12	0
1000	5	1000000	0	0.3825	0.2125	0	0.255	0.15	0
1000	5	1000000	0	0.369	0.205	0	0.246	0.18	0
1000	5	1000000	0	0.3555	0.1975	0	0.237	0.21	0
1000	5	1000000	0	0.342	0.19	0	0.228	0.24	0
1000	5	1000000	0	0.3285	0.1825	0	0.219	0.27	0
1000	5	1000000	0	0.315	0.175	0	0.21	0.3	0

(c) Run 1 Group 2

1000	5	1000000	0	0.315	0.175	0	0.21	0	0.3
1000	5	1000000	0	0.315	0.175	0	0.21	0.03	0.27
1000	5	1000000	0	0.315	0.175	0	0.21	0.06	0.24
1000	5	1000000	0	0.315	0.175	0	0.21	0.09	0.21
1000	5	1000000	0	0.315	0.175	0	0.21	0.12	0.18
1000	5	1000000	0	0.315	0.175	0	0.21	0.15	0.15
1000	5	1000000	0	0.315	0.175	0	0.21	0.18	0.12
1000	5	1000000	0	0.315	0.175	0	0.21	0.21	0.09
1000	5	1000000	0	0.315	0.175	0	0.21	0.24	0.06
1000	5	1000000	0	0.315	0.175	0	0.21	0.27	0.03
1000	5	1000000	0	0.315	0.175	0	0.21	0.3	0

(d) Run 2 Group 2

1000	5	1000000	0.1	0.1	0.1	0.7	0	0	0
1000	5	1000000	0.097	0.097	0.097	0.679	0	0.03	0
1000	5	1000000	0.094	0.094	0.094	0.658	0	0.06	0
1000	5	1000000	0.091	0.091	0.091	0.637	0	0.09	0
1000	5	1000000	0.088	0.088	0.088	0.616	0	0.12	0
1000	5	1000000	0.085	0.085	0.085	0.595	0	0.15	0
1000	5	1000000	0.082	0.082	0.082	0.574	0	0.18	0
1000	5	1000000	0.079	0.079	0.079	0.553	0	0.21	0
1000	5	1000000	0.076	0.076	0.076	0.532	0	0.24	0
1000	5	1000000	0.073	0.073	0.073	0.511	0	0.27	0
1000	5	1000000	0.07	0.07	0.07	0.49	0	0.3	0

(e) Run 1 Group 3

1000	5	1000000	0.07	0.07	0.07	0.49	0	0	0.3
1000	5	1000000	0.07	0.07	0.07	0.49	0	0.03	0.27
1000	5	1000000	0.07	0.07	0.07	0.49	0	0.06	0.24
1000	5	1000000	0.07	0.07	0.07	0.49	0	0.09	0.21
1000	5	1000000	0.07	0.07	0.07	0.49	0	0.12	0.18
1000	5	1000000	0.07	0.07	0.07	0.49	0	0.15	0.15
1000	5	1000000	0.07	0.07	0.07	0.49	0	0.18	0.12
1000	5	1000000	0.07	0.07	0.07	0.49	0	0.21	0.09
1000	5	1000000	0.07	0.07	0.07	0.49	0	0.24	0.06
1000	5	1000000	0.07	0.07	0.07	0.49	0	0.27	0.03
1000	5	1000000	0.07	0.07	0.07	0.49	0	0.3	0

(f) Run 2 Group 3

1000	5	1000000	0.3	0.3	0.3	0.1	0	0	0
1000	5	1000000	0.291	0.291	0.291	0.097	0	0.03	0
1000	5	1000000	0.282	0.282	0.282	0.094	0	0.06	0
1000	5	1000000	0.273	0.273	0.273	0.091	0	0.09	0
1000	5	1000000	0.264	0.264	0.264	0.088	0	0.12	0
1000	5	1000000	0.255	0.255	0.255	0.085	0	0.15	0
1000	5	1000000	0.246	0.246	0.246	0.082	0	0.18	0
1000	5	1000000	0.237	0.237	0.237	0.079	0	0.21	0
1000	5	1000000	0.228	0.228	0.228	0.076	0	0.24	0
1000	5	1000000	0.219	0.219	0.219	0.073	0	0.27	0
1000	5	1000000	0.21	0.21	0.21	0.07	0	0.3	0

(g) Run 1 Group 4

1000	5	1000000	0.21	0.21	0.21	0.07	0	0	0.3
1000	5	1000000	0.21	0.21	0.21	0.07	0	0.03	0.27
1000	5	1000000	0.21	0.21	0.21	0.07	0	0.06	0.24
1000	5	1000000	0.21	0.21	0.21	0.07	0	0.09	0.21
1000	5	1000000	0.21	0.21	0.21	0.07	0	0.12	0.18
1000	5	1000000	0.21	0.21	0.21	0.07	0	0.15	0.15
1000	5	1000000	0.21	0.21	0.21	0.07	0	0.18	0.12
1000	5	1000000	0.21	0.21	0.21	0.07	0	0.21	0.09
1000	5	1000000	0.21	0.21	0.21	0.07	0	0.24	0.06
1000	5	1000000	0.21	0.21	0.21	0.07	0	0.27	0.03
1000	5	1000000	0.21	0.21	0.21	0.07	0	0.3	0

(h) Run 2 Group 4

1000	5	1000000	0.15	0.4	0.3	0.15	0	0	0
1000	5	1000000	0.1455	0.388	0.291	0.1455	0	0.03	0
1000	5	1000000	0.141	0.376	0.282	0.141	0	0.06	0
1000	5	1000000	0.1365	0.364	0.273	0.1365	0	0.09	0
1000	5	1000000	0.132	0.352	0.264	0.132	0	0.12	0
1000	5	1000000	0.1275	0.34	0.255	0.1275	0	0.15	0
1000	5	1000000	0.123	0.328	0.246	0.123	0	0.18	0
1000	5	1000000	0.1185	0.316	0.237	0.1185	0	0.21	0
1000	5	1000000	0.114	0.304	0.228	0.114	0	0.24	0
1000	5	1000000	0.1095	0.292	0.219	0.1095	0	0.27	0
1000	5	1000000	0.105	0.28	0.21	0.105	0	0.3	0

(i) Run 1 Group 5

1000	5	1000000	0.105	0.28	0.21	0.105	0	0	0.3
1000	5	1000000	0.105	0.28	0.21	0.105	0	0.03	0.27
1000	5	1000000	0.105	0.28	0.21	0.105	0	0.06	0.24
1000	5	1000000	0.105	0.28	0.21	0.105	0	0.09	0.21
1000	5	1000000	0.105	0.28	0.21	0.105	0	0.12	0.18
1000	5	1000000	0.105	0.28	0.21	0.105	0	0.15	0.15
1000	5	1000000	0.105	0.28	0.21	0.105	0	0.18	0.12
1000	5	1000000	0.105	0.28	0.21	0.105	0	0.21	0.09
1000	5	1000000	0.105	0.28	0.21	0.105	0	0.24	0.06
1000	5	1000000	0.105	0.28	0.21	0.105	0	0.27	0.03
1000	5	1000000	0.105	0.28	0.21	0.105	0	0.3	0

(j) Run 2 Group 5

1000	5	1000000	0.2	0.1	0.5	0.2	0	0	0
1000	5	1000000	0.194	0.097	0.485	0.194	0	0.03	0
1000	5	1000000	0.188	0.094	0.47	0.188	0	0.06	0
1000	5	1000000	0.182	0.091	0.455	0.182	0	0.09	0
1000	5	1000000	0.176	0.088	0.44	0.176	0	0.12	0
1000	5	1000000	0.17	0.085	0.425	0.17	0	0.15	0
1000	5	1000000	0.164	0.082	0.41	0.164	0	0.18	0
1000	5	1000000	0.158	0.079	0.395	0.158	0	0.21	0
1000	5	1000000	0.152	0.076	0.38	0.152	0	0.24	0
1000	5	1000000	0.146	0.073	0.365	0.146	0	0.27	0
1000	5	1000000	0.14	0.07	0.35	0.14	0	0.3	0

(k) Run 1 Group 6

1000	5	1000000	0.14	0.07	0.35	0.14	0	0	0.3
1000	5	1000000	0.14	0.07	0.35	0.14	0	0.03	0.27
1000	5	1000000	0.14	0.07	0.35	0.14	0	0.06	0.24
1000	5	1000000	0.14	0.07	0.35	0.14	0	0.09	0.21
1000	5	1000000	0.14	0.07	0.35	0.14	0	0.12	0.18
1000	5	1000000	0.14	0.07	0.35	0.14	0	0.15	0.15
1000	5	1000000	0.14	0.07	0.35	0.14	0	0.18	0.12
1000	5	1000000	0.14	0.07	0.35	0.14	0	0.21	0.09
1000	5	1000000	0.14	0.07	0.35	0.14	0	0.24	0.06
1000	5	1000000	0.14	0.07	0.35	0.14	0	0.27	0.03
1000	5	1000000	0.14	0.07	0.35	0.14	0	0.3	0

(l) Run 2 Group 6

1000	5	1000000	0.25	0.25	0.25	0.25	0	0	0
1000	5	1000000	0.2425	0.2425	0.2425	0.2425	0	0.03	0
1000	5	1000000	0.235	0.235	0.235	0.235	0	0.06	0
1000	5	1000000	0.2275	0.2275	0.2275	0.2275	0	0.09	0
1000	5	1000000	0.22	0.22	0.22	0.22	0	0.12	0
1000	5	1000000	0.2125	0.2125	0.2125	0.2125	0	0.15	0
1000	5	1000000	0.205	0.205	0.205	0.205	0	0.18	0
1000	5	1000000	0.1975	0.1975	0.1975	0.1975	0	0.21	0
1000	5	1000000	0.19	0.19	0.19	0.19	0	0.24	0
1000	5	1000000	0.1825	0.1825	0.1825	0.1825	0	0.27	0
1000	5	1000000	0.175	0.175	0.175	0.175	0	0.3	0

(m) Run 1 Group 7

1000	5	1000000	0.175	0.175	0.175	0.175	0	0	0.3
1000	5	1000000	0.175	0.175	0.175	0.175	0	0.03	0.27
1000	5	1000000	0.175	0.175	0.175	0.175	0	0.06	0.24
1000	5	1000000	0.175	0.175	0.175	0.175	0	0.09	0.21
1000	5	1000000	0.175	0.175	0.175	0.175	0	0.12	0.18
1000	5	1000000	0.175	0.175	0.175	0.175	0	0.15	0.15
1000	5	1000000	0.175	0.175	0.175	0.175	0	0.18	0.12
1000	5	1000000	0.175	0.175	0.175	0.175	0	0.21	0.09
1000	5	1000000	0.175	0.175	0.175	0.175	0	0.24	0.06
1000	5	1000000	0.175	0.175	0.175	0.175	0	0.27	0.03
1000	5	1000000	0.175	0.175	0.175	0.175	0	0.3	0

(n) Run 2 Group 7

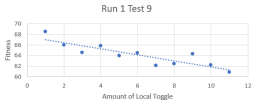
1000	5	1000000	0.2	0.2	0.2	0.4	0	0	0
1000	5	1000000	0.194	0.194	0.194	0.388	0	0.03	0
1000	5	1000000	0.188	0.188	0.188	0.376	0	0.06	0
1000	5	1000000	0.182	0.182	0.182	0.364	0	0.09	0
1000	5	1000000	0.176	0.176	0.176	0.352	0	0.12	0
1000	5	1000000	0.17	0.17	0.17	0.34	0	0.15	0
1000	5	1000000	0.164	0.164	0.164	0.328	0	0.18	0
1000	5	1000000	0.158	0.158	0.158	0.316	0	0.21	0
1000	5	1000000	0.152	0.152	0.152	0.304	0	0.24	0
1000	5	1000000	0.146	0.146	0.146	0.292	0	0.27	0
1000	5	1000000	0.14	0.14	0.14	0.28	0	0.3	0

(o) Run 1 Group 8

1000	5	1000000	0.14	0.14	0.14	0.28	0	0	0.3
1000	5	1000000	0.14	0.14	0.14	0.28	0	0.03	0.27
1000	5	1000000	0.14	0.14	0.14	0.28	0	0.06	0.24
1000	5	1000000	0.14	0.14	0.14	0.28	0	0.09	0.21
1000	5	1000000	0.14	0.14	0.14	0.28	0	0.12	0.18
1000	5	1000000	0.14	0.14	0.14	0.28	0	0.15	0.15
1000	5	1000000	0.14	0.14	0.14	0.28	0	0.18	0.12
1000	5	1000000	0.14	0.14	0.14	0.28	0	0.21	0.09
1000	5	1000000	0.14	0.14	0.14	0.28	0	0.24	0.06
1000	5	1000000	0.14	0.14	0.14	0.28	0	0.27	0.03
1000	5	1000000	0.14	0.14	0.14	0.28	0	0.3	0

(p) Run 2 Group 8

1000	5	1000000	0.3	0.1	0.3	0.3	0	0	0
1000	5	1000000	0.291	0.097	0.291	0.291	0	0.03	0
1000	5	1000000	0.282	0.094	0.282	0.282	0	0.06	0
1000	5	1000000	0.273	0.091	0.273	0.273	0	0.09	0
1000	5	1000000	0.264	0.088	0.264	0.264	0	0.12	0
1000	5	1000000	0.255	0.085	0.255	0.255	0	0.15	0
1000	5	1000000	0.246	0.082	0.246	0.246	0	0.18	0
1000	5	1000000	0.237	0.079	0.237	0.237	0	0.21	0
1000	5	1000000	0.228	0.076	0.228	0.228	0	0.24	0
1000	5	1000000	0.219	0.073	0.219	0.219	0	0.27	0
1000	5	1000000	0.21	0.07	0.21	0.21	0	0.3	0



(q) Run 1 Group 9

1000	5	1000000	0.21	0.07	0.21	0.21	0	0	0.3
1000	5	1000000	0.21	0.07	0.21	0.21	0	0.03	0.27
1000	5	1000000	0.21	0.07	0.21	0.21	0	0.06	0.24
1000	5	1000000	0.21	0.07	0.21	0.21	0	0.09	0.21
1000	5	1000000	0.21	0.07	0.21	0.21	0	0.12	0.18
1000	5	1000000	0.21	0.07	0.21	0.21	0	0.15	0.15
1000	5	1000000	0.21	0.07	0.21	0.21	0	0.18	0.12
1000	5	1000000	0.21	0.07	0.21	0.21	0	0.21	0.09
1000	5	1000000	0.21	0.07	0.21	0.21	0	0.24	0.06
1000	5	1000000	0.21	0.07	0.21	0.21	0	0.27	0.03
1000	5	1000000	0.21	0.07	0.21	0.21	0	0.3	0

(r) Run 2 Group 9

1000	5	1000000	0.15	0.35	0.35	0.15	0	0	0
1000	5	1000000	0.1455	0.3395	0.3395	0.1455	0	0.03	0
1000	5	1000000	0.141	0.329	0.329	0.141	0	0.06	0
1000	5	1000000	0.1365	0.3185	0.3185	0.1365	0	0.09	0
1000	5	1000000	0.132	0.308	0.308	0.132	0	0.12	0
1000	5	1000000	0.1275	0.2975	0.2975	0.1275	0	0.15	0
1000	5	1000000	0.123	0.287	0.287	0.123	0	0.18	0
1000	5	1000000	0.1185	0.2765	0.2765	0.1185	0	0.21	0
1000	5	1000000	0.114	0.266	0.266	0.114	0	0.24	0
1000	5	1000000	0.1095	0.2555	0.2555	0.1095	0	0.27	0
1000	5	1000000	0.105	0.245	0.245	0.105	0	0.3	0

(s) Run 1 Group 10

1000	5	1000000	0.105	0.245	0.245	0.105	0	0	0.3
1000	5	1000000	0.105	0.245	0.245	0.105	0	0.03	0.27
1000	5	1000000	0.105	0.245	0.245	0.105	0	0.06	0.24
1000	5	1000000	0.105	0.245	0.245	0.105	0	0.09	0.21
1000	5	1000000	0.105	0.245	0.245	0.105	0	0.12	0.18
1000	5	1000000	0.105	0.245	0.245	0.105	0	0.15	0.15
1000	5	1000000	0.105	0.245	0.245	0.105	0	0.18	0.12
1000	5	1000000	0.105	0.245	0.245	0.105	0	0.21	0.09
1000	5	1000000	0.105	0.245	0.245	0.105	0	0.24	0.06
1000	5	1000000	0.105	0.245	0.245	0.105	0	0.27	0.03
1000	5	1000000	0.105	0.245	0.245	0.105	0	0.3	0

(t) Run 2 Group 10

1000	5	1000000	0	0	0	0	0	0.1	0.9
1000	5	1000000	0	0	0	0	0	0.2	0.8
1000	5	1000000	0	0	0	0	0	0.3	0.7
1000	5	1000000	0	0	0	0	0	0.4	0.6
1000	5	1000000	0	0	0	0	0	0.5	0.5
1000	5	1000000	0	0	0	0	0	0.6	0.4
1000	5	1000000	0	0	0	0	0	0.7	0.3
1000	5	1000000	0	0	0	0	0	0.8	0.2
1000	5	1000000	0	0	0	0	0	0.9	0.1
1000	5	1000000	0	0	0	0	0	1	0

(u) Null Experiment

Figure B.6: Local Toggle's Experimental Settings

B.7 Add/Delete Experiments

1000	5	1000000	0.2	0.2	0	0.3	0	0.2	0.1
1000	5	1000000	0.2	0.2	0.03	0.27	0	0.2	0.1
1000	5	1000000	0.2	0.2	0.03	0.24	0	0.2	0.1
1000	5	1000000	0.2	0.2	0.09	0.21	0	0.2	0.1
1000	5	1000000	0.2	0.2	0.12	0.18	0	0.2	0.1
1000	5	1000000	0.2	0.2	0.15	0.15	0	0.2	0.1
1000	5	1000000	0.2	0.2	0.18	0.12	0	0.2	0.1
1000	5	1000000	0.2	0.2	0.21	0.09	0	0.2	0.1
1000	5	1000000	0.2	0.2	0.24	0.06	0	0.2	0.1
1000	5	1000000	0.2	0.2	0.27	0.03	0	0.2	0.1
1000	5	1000000	0.2	0.2	0.3	0	0	0.2	0.1

(a) Group 1

1000	5	1000000	0.2	0.2	0	0.3	0	0.2	0.1
1000	5	1000000	0.2	0.2	0.03	0.27	0	0.2	0.1
1000	5	1000000	0.2	0.2	0.03	0.24	0	0.2	0.1
1000	5	1000000	0.2	0.2	0.09	0.21	0	0.2	0.1
1000	5	1000000	0.2	0.2	0.12	0.18	0	0.2	0.1
1000	5	1000000	0.2	0.2	0.15	0.15	0	0.2	0.1
1000	5	1000000	0.2	0.2	0.18	0.12	0	0.2	0.1
1000	5	1000000	0.2	0.2	0.21	0.09	0	0.2	0.1
1000	5	1000000	0.2	0.2	0.24	0.06	0	0.2	0.1
1000	5	1000000	0.2	0.2	0.27	0.03	0	0.2	0.1
1000	5	1000000	0.2	0.2	0.3	0	0	0.2	0.1

(b) Group 2

1000	5	1000000	0.2	0.2	0	0.3	0	0.2	0.1
1000	5	1000000	0.2	0.2	0.03	0.27	0	0.2	0.1
1000	5	1000000	0.2	0.2	0.03	0.24	0	0.2	0.1
1000	5	1000000	0.2	0.2	0.09	0.21	0	0.2	0.1
1000	5	1000000	0.2	0.2	0.12	0.18	0	0.2	0.1
1000	5	1000000	0.2	0.2	0.15	0.15	0	0.2	0.1
1000	5	1000000	0.2	0.2	0.18	0.12	0	0.2	0.1
1000	5	1000000	0.2	0.2	0.21	0.09	0	0.2	0.1
1000	5	1000000	0.2	0.2	0.24	0.06	0	0.2	0.1
1000	5	1000000	0.2	0.2	0.27	0.03	0	0.2	0.1
1000	5	1000000	0.2	0.2	0.3	0	0	0.2	0.1

(c) Group 3

1000	5	1000000	0.2	0.2	0	0.3	0	0.2	0.1
1000	5	1000000	0.2	0.2	0.03	0.27	0	0.2	0.1
1000	5	1000000	0.2	0.2	0.03	0.24	0	0.2	0.1
1000	5	1000000	0.2	0.2	0.09	0.21	0	0.2	0.1
1000	5	1000000	0.2	0.2	0.12	0.18	0	0.2	0.1
1000	5	1000000	0.2	0.2	0.15	0.15	0	0.2	0.1
1000	5	1000000	0.2	0.2	0.18	0.12	0	0.2	0.1
1000	5	1000000	0.2	0.2	0.21	0.09	0	0.2	0.1
1000	5	1000000	0.2	0.2	0.24	0.06	0	0.2	0.1
1000	5	1000000	0.2	0.2	0.27	0.03	0	0.2	0.1
1000	5	1000000	0.2	0.2	0.3	0	0	0.2	0.1

(d) Group 4

1000	5	1000000	0.2	0.2	0	0.3	0	0.2	0.1
1000	5	1000000	0.2	0.2	0.03	0.27	0	0.2	0.1
1000	5	1000000	0.2	0.2	0.03	0.24	0	0.2	0.1
1000	5	1000000	0.2	0.2	0.09	0.21	0	0.2	0.1
1000	5	1000000	0.2	0.2	0.12	0.18	0	0.2	0.1
1000	5	1000000	0.2	0.2	0.15	0.15	0	0.2	0.1
1000	5	1000000	0.2	0.2	0.18	0.12	0	0.2	0.1
1000	5	1000000	0.2	0.2	0.21	0.09	0	0.2	0.1
1000	5	1000000	0.2	0.2	0.24	0.06	0	0.2	0.1
1000	5	1000000	0.2	0.2	0.27	0.03	0	0.2	0.1
1000	5	1000000	0.2	0.2	0.3	0	0	0.2	0.1

(e) Group 5

1000	5	1000000	0.2	0.2	0	0.3	0	0.2	0.1
1000	5	1000000	0.2	0.2	0.03	0.27	0	0.2	0.1
1000	5	1000000	0.2	0.2	0.03	0.24	0	0.2	0.1
1000	5	1000000	0.2	0.2	0.09	0.21	0	0.2	0.1
1000	5	1000000	0.2	0.2	0.12	0.18	0	0.2	0.1
1000	5	1000000	0.2	0.2	0.15	0.15	0	0.2	0.1
1000	5	1000000	0.2	0.2	0.18	0.12	0	0.2	0.1
1000	5	1000000	0.2	0.2	0.21	0.09	0	0.2	0.1
1000	5	1000000	0.2	0.2	0.24	0.06	0	0.2	0.1
1000	5	1000000	0.2	0.2	0.27	0.03	0	0.2	0.1
1000	5	1000000	0.2	0.2	0.3	0	0	0.2	0.1

(f) Group 6

1000	5	1000000	0.2	0.2	0	0.3	0	0.2	0.1
1000	5	1000000	0.2	0.2	0.03	0.27	0	0.2	0.1
1000	5	1000000	0.2	0.2	0.03	0.24	0	0.2	0.1
1000	5	1000000	0.2	0.2	0.09	0.21	0	0.2	0.1
1000	5	1000000	0.2	0.2	0.12	0.18	0	0.2	0.1
1000	5	1000000	0.2	0.2	0.15	0.15	0	0.2	0.1
1000	5	1000000	0.2	0.2	0.18	0.12	0	0.2	0.1
1000	5	1000000	0.2	0.2	0.21	0.09	0	0.2	0.1
1000	5	1000000	0.2	0.2	0.24	0.06	0	0.2	0.1
1000	5	1000000	0.2	0.2	0.27	0.03	0	0.2	0.1
1000	5	1000000	0.2	0.2	0.3	0	0	0.2	0.1

(g) Group 7

1000	5	1000000	0.2	0.2	0	0.3	0	0.2	0.1
1000	5	1000000	0.2	0.2	0.03	0.27	0	0.2	0.1
1000	5	1000000	0.2	0.2	0.03	0.24	0	0.2	0.1
1000	5	1000000	0.2	0.2	0.09	0.21	0	0.2	0.1
1000	5	1000000	0.2	0.2	0.12	0.18	0	0.2	0.1
1000	5	1000000	0.2	0.2	0.15	0.15	0	0.2	0.1
1000	5	1000000	0.2	0.2	0.18	0.12	0	0.2	0.1
1000	5	1000000	0.2	0.2	0.21	0.09	0	0.2	0.1
1000	5	1000000	0.2	0.2	0.24	0.06	0	0.2	0.1
1000	5	1000000	0.2	0.2	0.27	0.03	0	0.2	0.1
1000	5	1000000	0.2	0.2	0.3	0	0	0.2	0.1

(h) Group 8

Figure B.7: Add/Delete's Experimental Settings