The Application of Adaptive Genetic Algorithms to Finding Ramsey Graphs

Sam Coy

Third Year Project

Supervised by
Prof. Artur Czumaj

April 25, 2018

Abstract

Studied for decades, Ramsey numbers have proven very difficult to determine exact values for, with only 11 found to date. Recently, search techniques have proven useful in improving lower bounds for Ramsey numbers, one example being genetic algorithms. This project concerns the application of parameter control techniques to this approach – a collection of techniques used to determine certain parameters in a genetic algorithm such as population size, crossover rates, and mutation probability, during the execution of the algorithm rather than beforehand. After conducting a literature review of both the application of search techniques to Ramsey's theorem and parameter control, this project examines the effect of two different parameter control approaches applied to the problem of finding Ramsey numbers. This project then compares these approaches, draws conclusions about their efficacy in this problem domain, and suggest avenues for further research in finding Ramsey numbers using search techniques.

Contents

1 Background

This project is concerned with the application of genetic algorithm techniques (specifically parameter control) to finding colourings of graphs under certain constraints. This section provides some context for the project and explains some of the challenges inherent to it.

In this section and this project as a whole, all graphs are undirected, and contain no multiple edges or self-loops. K_n is the "complete graph" of order n, that is n vertices with all pairs of vertices connected by an edge.

1.1 Ramsey Theory

Broadly, Ramsey Theory is a branch of combinatorics concerned with the conditions under which order must appear in mathematical strucutres. This project is concerned with Ramsey's Theorem, a theorem within Ramsey Theory concerned with finding monochromatic graphs within colourings of large complete graphs.

Specifically, the theorem defines the *Ramsey number* $R(r, s)$ such that $K_{R(r,s)}$ is the smallest complete graph for which any colouring of its edges using two colours, say red and blue, must contain a K_r where all internal edges are red, or a K_s where all internal edges are blue. Figure 1 shows a K_5 coloured in such a way. The colouring shown contains no red or blue K_3, and this example is maximal – any larger complete graphs cannot be coloured in such a way – therefore $R(3, 3) = 6$. We call a colouring of K_n with no red K_r or blue K_s an (r, s)-*colouring* of K_n. By symmetry, $R(r, s) = R(s, r)$.

Ramsey numbers are "non-trivial" if $r, s > 3$. If this condition is not satisfied the number is easy to calculate, as K_2 is simply a line; $R(2, n) = n$, as there cannot be any red edges whatsoever without creating a red K_2. A simple recurrent bound for $R(r, s)$ is given by:

$$R(r, s) \leq R(r - 1, s) + R(r, s - 1)$$

To see why this is true, consider the complete graph with $R(r - 1, s) + R(r, s - 1) + 1$ vertices. Choose a vertex v arbitrarily, and partition the other vertices into two sets: R if the edge connecting them to v is red, and B if that edge is blue. Either $|R| > R(r - 1, s)$, or $|B| > R(r, s - 1)$. If the former is true, then R contains a blue K_s, or a red K_{r-1} (which, with the inclusion of v forms a red K_r). If the latter is true then B must contain a red K_r, or a blue K_{s-1} (a blue K_s with the inclusion of v). Therefore in either case, the complete graph under consideration necessarily contains a red K_r or a blue K_s, and so $R(r, s) < R(r - 1, s) + R(r, s - 1) + 1$.

The theorem can be extended to colourings with more colours, and some of these cases

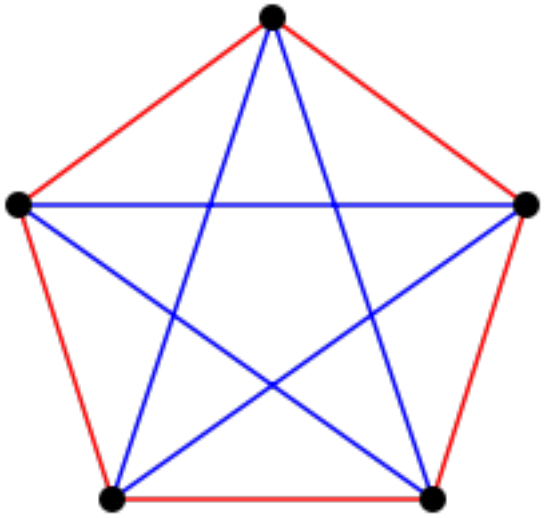

Figure 1: A colouring of K_5 with no red or blue K_3, cf. Richtom80 at English Wikipedia

will be examined later in this report. It has been shown that $R(r_1, r_2, \ldots, r_n)$ (the smallest size of complete graph that cannot be $(r_1, r_2, \ldots, r_n)$-coloured) exists and is finite, for all finite sequences of positive integers $(r_1, r_2, \ldots, r_n)$, but the value is notoriously challenging to find for even short sequences with small elements – only 9 non-trivial 2-colour Ramsey Numbers have been found so far, and only 2 non-trivial 3-colour Ramsey Numbers are known [7].

It is also possible to extend the theorem to include different substructures which any colouring must avoid, such as $K_n - e$ (K_n missing an edge), C_n (an n-length cycle), $K_{m,n}$ (complete bipartite graphs), and so on. These extensions form a broad field of study but are beyond the scope of this report. Similar techniques are used in search approaches to finding colourings without these alternative substructures as are used in finding colourings without monochromatic K_n [8], so it is likely that the conclusions of this project also apply to such problems.

1.1.1 Complexity

The cause of the remarkable difficulty of finding values of Ramsey Numbers is the search space. To show that $R(r, s) > n$, one has to find a colouring of K_n that contains no red K_r or blue K_s. Unfortunately, there are $2^{\binom{n}{2}}$ ways of colouring K_n, as there are two ways of colouring each edge, and K_n has $\binom{n}{2}$ edges. For example, one of the most prominent Ramsey numbers which has not yet been found is $R(5, 5)$, which has a lower bound of 43. There are $2^{\binom{43}{2}} = 2^{903}$ possible colourings of K_{43}, which is unfeasible for an exhaustive search, even accounting for isomorphisms in these colourings.

The problem of large search spaces only applies when finding lower bounds of Ramsey numbers, where finding valid colourings are how these bounds are demonstrated and advanced. Upper bounds to these numbers are generally found through mathematical arguments, usually involving linear programming or similar combinatorial techniques. This project is only concerned with techniques for finding lower bounds of Ramsey numbers.

1.1.2 Search Space

The space of colourings of complete graphs has several properties which make some search approaches harder to execute. Firstly, the space is discrete, and highly multidimensional, the function from a 2-colouring of K_n to the number of cliques in a colouring is defined over $0, 1^{\binom{n}{2}} \rightarrow \mathbb{N}$, which is a very unusual search space, which many search techniques are not designed to operate in. The difficulty comes from the domain, incremental change can only be achieved by changing a 1 to a 0 or vice versa, there is no more nuanced way to modify the colouring.

Another feature of the space of colourings of any given complete graph is that it is extremely multimodal and contains many "spikes", when the constraints of the colouring are tight enough. This means that small changes to a good solution in the space (one with a low fitness value), yield solutions of much lower quality. This can cause problems for many search heuristics, which can tend to converge to local optima as a result of this property. For example, consider the $(3, 3, 3)$-colouring of K_{16} in figure 2. This colouring is tight to its constraints: changing the colour of any one of its 120 edges creates two K_3s in that edge's new colour. This instability is particularly exacerbated for larger problem instances and instances using many colours.

This solution space is also not dense, especially as the size of graph under consideration approaches the value of the Ramsey number. For example, there are 2 distinct (non-isomorphic) $(4, 4)$-colourings of K_{16}, and only 1 distinct $(4, 4)$colouring of K_{17}, where $R(4, 4) = 18$ [9]. While after accounting for isomorphisms this number is significantly larger, it remains a relatively small proportion of the search space.

1.2 Genetic Algorithms

Genetic Algorithms are a search metaheuristic inspired by the process of natural selection. The search operates using a population of individuals, where each individual is a genome representation of a candidate solution in the search. A fitness function is defined, which is a measure of the quality of an individual, and evolutionary operators operate on individuals in the population from one generation to the next, with the hope of improving them. The

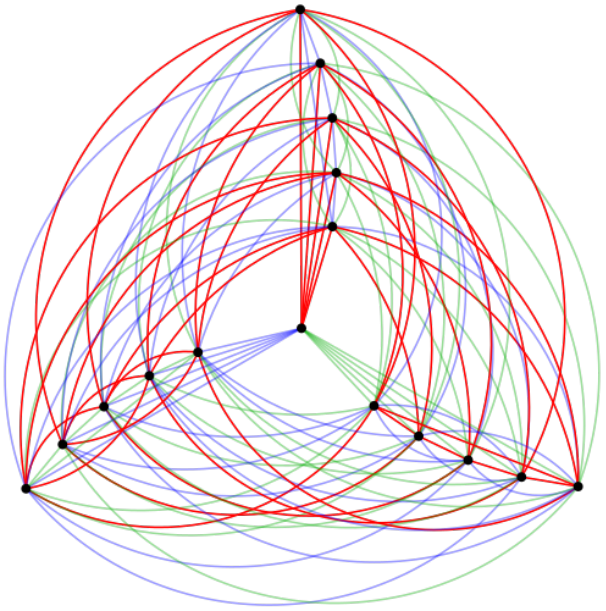

Figure 2: A $(3, 3, 3)$-colouring of K_{16}, cf. Maproom at English Wikipedia

basic structure of a genetic algorithm is as follows:

- The population is randomly initialised.
- While an optimal solution has not yet been found:
  - Every individual in the population is evaluated using the fitness function.
  - The best individuals in the population are selected to produce the next generation.
  - The next generation is produced using these individuals and the chosen evolutionary operators.

The two most widely used evolutionary operators are *crossover* (the combination of two "parent" individuals into a child), and *mutation* (random minor alteration of an individual). Other operators are possible but rarely used in practice.

Genetic Algorithms have proven to be very useful in several domains, such as scheduling problems. In general they prove effective in search spaces with a complex fitness landscape, as the operator of mutation is useful in avoiding the preemptive convergence of the population to a local optima.

1.2.1 Parameter Tuning vs Parameter Control

The operation of a genetic algorithm requires the setting of several parameters. In the case of the evolutionary operators this project is considering, the values p_c (the probability that

an individual will be selected for the crossover which produces the next generation), p_m (the probability that each gene in the genome will be mutated), and s (the population size) are to be determined. Finding good values for these parameters is extremely important; the performance of a genetic algorithm can range from random search to hill climbing depending on the parameters used.

There are two commonly used methods through which these values can be chosen [10]. The first is parameter tuning, and was the most widely used method during the inception and early development of genetic algorithms. This method consists of preselecting good values of crossover and mutation, usually determined by running small experiments beforehand to find values that work well. These values then remain constant for the duration of the algorithm's run. There have been efforts to make this manual parameter selection easier: De Jong ran experiments to determine values that performed well under many numeric problems [11], and other attempts have been made to provide values of these parameters that can be widely applied over specific sets of problem types. However, this practice is often criticised as a poor way of setting parameters, for two reasons: despite guidelines for certain problem types (often referred to as "setting parameters by analogy"), manually tuning the parameters can be extremely time-consuming; and it has been argued that fixing these parameters for the duration of a run is often not optimal, and that true optimal values for these parameters often vary during a run [12].

The alternative approach is parameter control, a more recent approach to parameter selection in genetic algorithms which, instead of preselecting parameters, selects and varies them at runtime based on the current state of the genetic algorithm. There are many measurable properties which can be used to vary the parameters, including the diversity or prosperity of the population at a given point, the current generation number, the rate of successful mutations in the recent past, and so on. Parameter control is a broad field of study and the properties and applications of the many ways to implement it are the subject of extensive research.

Parameter control seems to have several intrinsic advantages over the alternative: it takes far less time than selecting values manually and is far less prone to error, the parameters can change to become more efficient if the fitness landscape changes, and the parameters can take on more precise values. It can also be better suited to problems that have not been previously attempted using genetic algorithms and for which good values for parameters are not known, and similarly, problems for which the shape and properties of the fitness landscape is unknown.

1.3 Motivation

The aim of this project is to review surrounding literature in the area of search-based approaches to finding colourings of graphs in Ramsey's theorem, to review literature surrounding parameter control techniques, and to apply the parameter control techniquevarying the parameters based ons studied to the problem domain of Ramsey's theorem and compare the performance to traditional parameter tuning techniques.

This is a relatively novel project, in that while genetic algorithms have previously been applied in the search for valid colourings (although not particularly often, local search techniques are usually preferred), these approaches have primarily utilised parameter tuning techniques rather than parameter control techniques.

The use of parameter control techniques towards the end of finding valid (r, s)-colourings of graphs is certainly worth exploring: this domain has no broadly similar problems for which good values of the parameters are known, and there is reason to believe that the search space of the problem, as previously discussed, may benefit from the parameters changing over time, and therefore lends itself particularly well to parameter control techniques.

2 Literature Review

This section presents a brief overview of the papers analysed during the project's literature review which were most influential to the direction of this project.

2.1 J.G.Kalbfleisch: *Construction of Special Edge-Chromatic Graphs*, [1]

This paper is a typical example of how lower bounds can be constructed for Ramsey numbers by means of mathematical construction without the need for complex and computationally intensive search techniques [1]. While it is the use of search techniques that is the subject of the testing in this report, it is still important to understand the alternative approaches used. Being from 1965, this article generated some important results in a field that was, at the time, still in relative infancy.

The approach in this case was the introduction and use of *regular colourings*, in which every edge of the same length (where the graph's vertices are equally spaced on the circumference of a circle) is coloured the same colour. It turns out that in many cases, the maximal colourings for certain constraints, or at least graphs that are nearly maximal, are regular. The lower bounds $R(3, 7) \geq 22, R(3, 8) \geq 27, R(4, 5) \geq 25$ were constructed as a

result of the development of this technique. The former two are actually only one lower than the true value, and the bound for $R(4,5)$ was confirmed as the true value by McKay and Radziszowski [13].

Sadly, it seems to be the case that while certain patterns have proven useful for the discovery of certain smaller Ramsey numbers, none have proven general-purpose enough for use in currently challenging problem instances. Most modern approaches have used computer search because after a certain size, enumeration, even performed in a particularly clever manner using certain techniques to reduce the workload, becomes unfeasible.

2.2 G.Exoo: *On Some Small Classical Ramsey Numbers*, [2]

Exoo's methods and work have proved extremely valuable in finding Ramsey numbers, specifically the computational advances of lower bounds. This paper [2] improved the lower bounds of $R(3,11)$ and $R(4,8)$ by one each, using a constructive, local search approach.

While this project is more focused on global search, lessons can certainly be learned from the technique which Exoo uses. Exoo starts with a colouring randomly extended from a good colouring of a smaller graph. Then each edge is examined (in a random order), and a comparison of the two graphs resulting from the two possible colourings of this edge are examined, with the better colouring being used in the next edge examination with a fixed (high) probability; occasionally the worse of the two colourings are selected in order to prevent the technique getting stuck at local maxima.

There is an interesting detail that Exoo brings up regarding how solution quality is determined: he claims that for off-diagonal Ramsey numbers ($R(r,s)$ where $r \neq s$), illegal cliques should not be weighted the same; smaller cliques are inherently worse as they are harder to remove. No justification is given for this in the note besides "empirical evidence", but it certainly seems plausible when you consider that his approach checks (and therefore changes) edges at random, and larger cliques have more edges and so are more likely to change sooner. The exact weighting given is $w(K_s) = \left(\frac{r}{s}\right)^{3/2} w(K_r)$, where $w(K_s)$ is the weight given to a K_s (lower is better).

2.3 C.J.Kunkel, P.Ng: *Ramsey Numbers: Improving the Bounds of* $R(5,5)$, [3]

Kunkel and Ng attempted to apply genetic algorithm techniques to improve the lower bound of $R(5,5)$, which was 43 at the time [14, 3]. Their approach was not ultimately successful; 43 remains the lower bound to this day, and there is strong experimental evidence, given

and argued for by Radziszowski and MacKay, that $R(5,5) = 43$ [15].

Their approach to applying genetic algorithm techniques to the problem of finding valid colourings of Ramsey graphs was as follows:

- The genome of each colouring was represented by the flattened lower half of an adjacency matrix, where each entry is a "colour" (they used 0 and 1).
- The fitness of a colouring was defined as the number of monochromatic K_5 in the colouring, and the objective of the algorithm is to minimise the fitness (and ideally find a candidate with a fitness of 0).
- The evolutionary operators were:
  - **Crossover:** Crossover involves taking the genomes of two individuals (I_1, I_2)and merging them together, that is selecting a random index and creating a new genome consisting of the genome of I_1 before the index, and the genome of I_2 after the index.
  - **Mutation:** Mutation is an operation on a single individual, where each colour in the genome is changed with a small probability p_m.

They ran the algorithm to generation 400, 240 times. The best ten runs yielded a best individual with a fitness between 1300 and 1400, with the best candidate of the best run having a fitness of 1300. It is not explicitly mentioned in the text of the paper, but from their conclusion one can infer that they used parameter tuning techniques.

Many of their design decisions seem sensible and worth emulating; there are no obviously better ways to represent a graph colouring than the one they choose, and the fitness function is both simple and admissible. Their approach to mutation is also a good way of making minor alterations to a colouring by modifying individual genes. The implementation of crossover is questionable however; does single point crossover preserve the quality of the parents in the children? It is certainly a point worth considering, although the authors choose not to.

The results of their approach may have been limited in some respects by the hardware available at the time. With modern hardware, better results may be obtainable, although it is still hard to imagine that those results would yield a colouring. The design decisions made by Kunkel and Ng may have had a significantly bigger impact on the results, for example their choice of operators and implementation of those operators. Overall however, this is a good reference implementation for a genetic algorithm approach to Ramsey's theorem.

2.4 T.Bäck, M.Schütz: *Intelligent Mutation Rate Control in Canonical Genetic Algorithms*, [4]

This paper is one of several attempts to modify the rate of mutation in a genetic algorithm using a fixed schedule whereby its value is inversely proportional to the current time [4]. This approach to an adaptive rate of mutation is an obvious and sensible choice, and is an idea in which certain global search techniques, particularly simulated annealing, are rooted. In fact, this is alluded to in the paper, with simulated annealing being referred to as "a simple $(1+1)$ algorithm", interpreting it as a form of evolutionary algorithm with a population and successor size of 1.

In fact, they derive both a deterministically decreasing schedule as described, as well as a so-described "self-adaptation" method, however for the majority of problems over which the two were compared the deterministic schedule yielded improved results and will be the approach considered here. The precise schedule which they choose is:

$$p_t = \left(2 + \frac{n-2}{T-1} \cdot t\right)^{-1}$$

In the above equation, T represents the maximum generation number, while n represents the genome size and t represents the current generation. The result of this is an inversely proportional relationship between the current mutation rate and the time.

This approach would face challenges when implemented to produce valid colourings, however. There is no reasonable "maximum generation" to set – the problem is only solved when the fitness function reaches its optimal value, and not before. This could be disregarded and replaced with an arbitrarily high number, but the schedule relies to an extent on a certain end point. The improved performance does provide some justification for the project; they presented strong evidence that variable mutation provides better results for some problem types.

2.5 M.Srinivas, L.M.Patnaik: *Adaptive Probabilities of Crossover and Mutation in Genetic Algorithms*, [5]

In their paper, Srinivas and Patnaik outline an approach to modifying the rates of crossover and mutation, by varying these values with the distribution of the fitness of the population and the fitness of the individuals themselves [5]. This technique is relatively unusual in that it has individual values of crossover and mutation, rather than one for the entire population.

Their rationale behind this technique is that the better individuals of the population remain fairly stable from generation to generation, and the worse individuals are given larger

values of crossover and mutation and employed to search the local space. This, theoretically, should assist in very multimodal search spaces, such as the subject of this project, as this population cannot converge to a local optima as easily as genetic algorithms with static parameters can; the solutions that are employed to search the space have a much higher mutation and crossover rate than tuned parameters and are therefore easily able to leave local optima.

Specifically, they determine the parameters of p_c and p_m using the formulae:

$$p_c = k_1(f_{max} - f')(f_{max} - \bar{f}), f' \geq \bar{f}$$
$$p_c = k_3, f' < \bar{f}$$
$$p_m = k_2(f_{max} - f)(f_{max} - \bar{f}), f \geq \bar{f}$$
$$p_m = k_4, f < \bar{f}$$

In the above examples, k_1, k_2, k_3, k_4 are parameters themselves to be determined in some way. While it seems that this is simply deferring the problem of parameter selection, Srinivas and Patnaik argue that $k_1 = k_3 = 1.0$, and $k_2 = k_4 = 0.5$ are good general case values for these parameters. They also instate a value of "base mutation", pointing out that the population might need some disruption even for the very best individuals, and they assign a value of 0.0001 for this purpose. Note that this is used as a baseline for all members of the population, meaning that the best individual may actually decrease in fitness from one generation to the next; most genetic algorithm implementations do not mutate the individuals that were selected, only the offspring.

They offer a literature review of their own, comparing their methods to previous "adaptive" strategies for genetic algorithms. The greatest difference, it seems, is that their approach varies the parameters individually, and not according to a predetermined schedule. Most techniques for varying these parameters previously would modify the parameters for the entire population in a manner similar to simulated annealing.

Their results showed an improvement over static genetic algorithms in the cases on which they experiment (TSP, DeJong's f5, f6, f7 to name a few – all highly multimodal spaces), and go on to conclude about the scenarios under which the technique performs well, stating that "it appears that the AGA performs relatively better than the SGA when the number of local optima in the search space is large"[5, p.664]. The conclusion also mentions that the performance of the adaptive approach varies with the epistacity of the problem (how dependent the effect of one gene is on the presence of other genes, in a non-additive way). Both of these indicate that this approach may be well suited to the problem domain and is worth examining.

2.6 J.Lis, M.Lis: *Self-adapting parallel genetic algorithm with the dynamic mutation probability, crossover rate and population size*, [6]

In their paper, Lis and Lis describe a "parallel farming"-based approach to controlling values of crossover rate, mutation probability, and population size [6]. This is accomplished by training multiple populations (Lis and Lis use 9), with different values of these parameters, and comparing the performance of the populations periodically, an event they call an "epoch". The values of the parameters of all populations are then shifted towards the values of the best-performing population. All populations then take the best genomes from all of the populations combined, and continue the evolutionary algorithm with their new parameters.

In order to independently evaluate the various parameter "levels" with as few populations as possible, Lis and Lis employed a Latin Square approach (as shown in figure 3) to distributing the levels of the parameters, so only 9 populations are needed. Each value is assigned a *Low*, *Medium*, or *High* value of a given parameter, in such a way that for every pair of parameter-levels (e.g. a *High* value for parameter 1, and a *Low* value for parameter 3), exactly one population (in the given case, population 8) has it. Only 9 populations are required to ensure this. This approach does assume that the parameters are independent, but in genetic algorithms they usually are to at least some extent, and this is an assumption which also underlies parameter tuning.

experiment number	1	2	3	4	5	6	7	8	9
parameter 1	L	L	L	M	M	M	H	H	H
parameter 2	L	M	H	L	M	H	L	M	H
parameter 3	L	M	H	M	H	L	H	L	M

Figure 3: Lis and Lis's Latin Square experiment design

Lis and Lis argued, additionally, that a parallel approach to the problem of dynamically determining parameters for genetic algorithms is a natural one, "due to the increasing demands for the application of genetic algorithms to large and complex search spaces"[6, p.81], the point being that these systems would be effective in distributed computing models. In their paper, they use a master-slave model for running the algorithm across a number of computers: the master determines when the epochs occur, and manages the population and parameter changes at each epoch, while the slaves simply perform the algorithm in between epochs using the parameters and population they have been assigned.

Their paper did assume that performance of the genetic algorithm based on population

size, crossover probability and mutation probability was unimodal, but references an earlier paper that indicated that this assumption was justified.

They performed experiments on numeric function optimisation (which is a different class of problem to the one being studied here), and found that their technique performed at least as well as the techniques proposed by Goldberg and Arabas[16, 17].

Translating their approach to the problem domain at hand is straightforward, as the approach is better described as a meta-approach to genetic algorithm training, rather than relying on specific implementation details as Srinivas and Patnaik's approach does. That is, the actual evolutionary computation is identical to the parameter tuning model, the difference only being that this computation only happens for a small number of iterations at a time.

3 Project Approach and Design

This section provides a high-level overview of the approach to the implementation of the algorithms, as well as an explanation of some of the design decisions underlying the program.

3.1 Project Approach

From the literature review two techniques seem particularly worthy of deep examination and testing. Srinivas and Patnaik's approach to parameter control, that of individually modifying mutation and crossover likelihoods, could prove useful in a search approach to finding valid colourings. The algorithm seems to target search spaces such as the one in this problem domain well.

Similarly, the approach proposed by Lis and Lis is worthy of testing. While structurally very different to traditional genetic algorithms, which use one population, multiple population approaches allow for maintaining diverse populations, while testing alternate values for genetic parameters.

The task, therefore, is to develop software to find valid colourings using genetic algorithms, extend that software to use the adaptive techniques laid out by Srinivas and Patnaik, and Lis and Lis, and compare and analyse the performance of the various techniques on a range of problem instances.

3.2 Project Management

3.2.1 Software Methodology

This project was developed using an Agile approach. As research was being done parallel to the development of the software, its structure needed to remain flexible as changes had to be made at unexpected times. Indeed it was determined that Lis and Lis's approach to parameter control would be one of the techniques to be compared fairly late into the development.

In addition, genetic algorithms are particularly prone to errors, as they are non-deterministic and take a long time to test, and it can be difficult to determine what changes have caused problems in the convergence of populations. The codebase and overall design need to remain flexible in order to enable easy identification and solution of bugs. Therefore an agile approach was the most appropriate for this project.

3.2.2 Code Management

Git 2.7 was used for version control, with a `master` branch used for stable versions of the software that were used to run experiments, and a `develop` branch used for implementing features. The repository was backed up to two different remote servers on a regular basis.

3.2.3 Time Management

A schedule was agreed to during the initial planning phase of the project, and it was adhered to reasonably well throughout. There was some variance during sections of the project due to unexpected workloads from other modules, and some of the inbuilt flexible time did end up being used (about 2 weeks of the flexible time out of the 3 that were scheduled). The addition of the Lis schedule, relatively late into the project, was another cause of delay. Not initially a planned feature, the incorporation of Lis and Lis's technique is an extension of the project scope, so a delay, especially when more than sufficient time remains, is especially justified.

The research was originally planned to take place during a short section of the project at the beginning, but it was decided that given the volume of research involved that some aspects of the research should be performed parallel to the implementation.

Bugfixing delayed the timetable as well, especially in the time immediately after the initial implementation of the algorithm. However, other elements took less time than expected, the implementation of Srinivas and Patnaik's technique being one of them.

3.3 Program Design

The software was structured in a modular way, as to allow for modifications to particular components during an experiment without affecting the experiment flow. There were four primary modules for the project:

- `graph.py`: the module that contained the implementation of the Graph class and the monochromatic clique detection logic.
- `evolution.py`: the module that contained the logic for the genetic algorithm, the two parameter control schedules being used, as well as the threading logic for the fitness function.
- `logging.py`: responsible for logging the data from the experiments.
- `chart.py`: responsible for producing charts of the performance of the various parameter schedules in the experiments.

The idea behind this structure is that the entry point to running the evolutionary simulation would be `evolution.py`. Arguments could be passed to this script (which would invoke `graph.py` during execution. `chart.py` was a separate script used exclusively to chart the resulting log files.

4 Implementation

4.1 Implementation Details

This section contains details of the implementation of the genetic algorithm in the software. Many of these decisions are simply those used by Kunkel and Ng in their paper, but alternatives to these exist which merit consideration and discussion, so these decisions are justified and examined thoroughly regardless.

4.1.1 Graph Representation

The representation of the graph colourings was similar to that used by Kunkel and Ng in their 2003 paper [3]. The colourings were represented as the lower half of the adjacency matrix of the colouring, with each colour being given a small integer value.

There are a number of alternatives to this approach. Edges could be represented as a probability of being red, for example, which could have the advantage of making mutation a better and more nuanced operator. This approach may be difficult to represent as a bit

vector (the standard representation of genomes in genetic algorithms) however, especially for experiments with more than two colours.

It would also cause problems for the fitness function, which would need to be re-devised: how can one evaluate whether a given set of k nodes form a clique when it is not clear whether the edges are red or blue? There are certainly various ways to do it, the fitness function could be made continuous, and each collection of k nodes could, instead of contributing a value of 0 (indicating no clique) or 1 (indicating a clique), contribute a value equal to the probability of the nodes forming a clique. But whether this fitness function would be admissible or useful is hard to determine, and it has the additional complexity of it being far harder to determine whether a valid colouring has been found: no longer would a fitness value of 0 be the only indicator of one, as a valid colouring might have a non-zero fitness function (a small chance of not being a valid colouring), and the fitness function is no longer an admissible heuristic for the search.

4.1.2 Mutation

The implementation of the mutation operator is identical to Kunkel and Ng's paper, a diagram of which can be seen in figure 4. Each "gene" (which given the representation is the colour of one edge) is modified between each generation with a probability of p_m. It should be noted that this is the probability of the edge being recoloured and it could end up with the same colour; the mutation probability is effectively half of its given value on two colour problems, two thirds its given value for 3 colours, and so on.

Figure 4: A diagram of Kunkel and Ng's implementation of mutation, cf. [3]

There are alternatives to this approach as well, but none are nearly as good. If a continuous probability model was used to represent the colourings, as discussed above, the mutation could be more continuous in nature (with a small probability p_m, the value of an individual gene would change by a normally distributed value). A more continuous approach than the one presented in this project is the norm for genetic algorithms, but unfortunately it does not make much sense in this case unless the continuous approach to the genome is adopted.

4.1.3 Crossover

Crossover is implemented as single-point crossover on two genomes, chosen at random from the set of the genomes deemed fit enough to advance to the next generation. That is, a single point is selected, a child is then produced by taking the genome of one parent before that point and the genome from the other parent after that point. The parents are preserved. The proportion of genomes from each generation that both proceed to the next generation and are in the pool for crossover is given by the *crossover probability*, p_c. This implementataion was also used in Kunkel and Ng's paper, and is shown in figure 5.

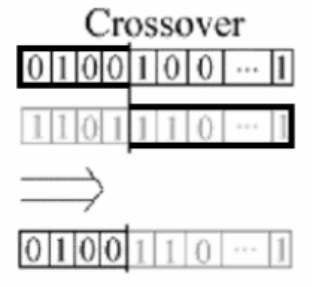

Figure 5: A diagram of Kunkel and Ng's implementation of crossover, cf. [3]

Alternative crossover techniques that are widely used in the domain of genetic algorithms are two-point crossover and uniform crossover. Two-point crossover consists of including the genome of the first parent between two randomly selected points in the genome, while uniform crossover consists of, at each position on the genome, a random selection from one of the parents. There is a considerable body of literature debating their relative merits in certain problem domains.

Differences also exist in the number of children produced by this operator, with some techniques preferring the "complement" of the child produced (the parts of the parent genomes that were not selected) to also be included in the next generation.

Ultimately, a single-point crossover was selected, as while other crossover techniques have shown promise, they performed differently under different circumstances, and for the purposes of control in the experiments, given that this project concerns the comparison of genetic algorithm techniques, a simple and widely useful technique was deemed appropriate.

4.1.4 Fitness Function

The fitness function used in this method is the number of cliques that violate the constraints of the problem instance. This approach is straightforward and an admissible heuristic.

Exoo [2] proposed that illegal cliques could contribute score to the fitness function inversely proportional to their size, for example an illegal K_3 would contribute twice as much score to the fitness function as an illegal K_6. The idea behind this is that smaller monochromatic cliques are harder to "break" than larger ones, and so are more concerning for the convergence of the algorithm, and there is some evidence that this produces better results under certain circumstances. However, for similar reasons to the crossover technique selection above, this may affect the different parameter control techniques differently, as it changes the shape of the fitness function, and so will not be employed for the sake of experimental control.

4.1.5 Logging

The logging mechanism for the software was fairly detailed. Different log formats were required for different purposes, for example logs of test runs designed to debug the software required all produced genomes to be logged, but doing this for all experiments would result in extremely large log files; a run $(4, 5)$-colouring K_{25} produced a 4GB file. For the experiments themselves, much less information was needed.

Therefore the level and detail of logging is configurable when running the software, from logging simply the best fitness in each generation to logging all of the parameter values and every single genome produced by the software.

4.2 Challenges

This section outlines some of the implementation challenges faced when developing the software, and how they were overcome.

4.2.1 Fitness Function Expense

The fitness function used in the implementation is, unfortunately, extremely expensive to evaluate ($O(n^k)$ complexity for counting the illegal k-cliques in a K_n). The basic fitness evaluation works how one would expect: each set of k nodes is examined, and if their internal edges are all the same, 1 is added to the score. Unfortunately, there are a lot of potential cliques to evaluate, especially on large graphs, and this must be done for each individual in each generation.

This is not necessarily a particularly severe problem: the aim of this project is not to find Ramsey numbers, but to evaluate several techniques. However, it does limit the amount of experimental data that can be collected, so it does pose a problem for the number and

quality of the experiments, and it would be desirable to have some way to speed up this function.

In response to this challenge, two improvements were made. Firstly, the function was multithreaded to allow for faster performance on multicore machines. This had s fairly small impact on the machine which was used to run the experiments, which has 2 cores, but it certainly provided some increase in speed.

Secondly, a "fuzzy fitness finder" was implemented. This approach recognises that it is not necessarily important to precisely evaluate the quality of poor solutions in the population; an approximate estimation suffices for most purposes. The finder works by stopping counting once it has determined that the probability that the fitness of the individual is better than double the fitness of the best individual in the population is below some threshold, and extrapolates the fitness given the cliques examined so far.

This fitness finder was generally a positive addition to the program. It did introduce some overhead, and in some preliminary experiments in development its addition negatively impacted the performance of the program. For this reason it can be switched off with a command line argument. There are also some minor losses in accuracy, the finder stops evaluating once it has determined the individual is of poor quality with a high probability, but with so many individuals being evaluated some are bound to be of better quality than the finder predicts. But this loss in accuracy is extremely negligible: running the same schedules with and without the fuzzy finder did not yield a significant difference in the speed of convergence relative to the number of generations.

4.2.2 Fitting Parameter Control Techniques to Problem Domain

The parameter control techniques presented are meant to be applicable to genetic algorithms in the general case. However there were some differences between the setup in this project and more standard genetic algorithms which caused translation difficulties.

The first challenge is the discrete nature of the problem. Most genetic algorithms have a continuous fitness function, and a continuous set of values that the genes could take, which helps in their operation greatly: it is possible to observe some kind of gradient of the function, and the genes can be modified to a variable degree. However in this case specifically, that is not possible given the selected representation, and so translating the approaches laid out in the papers, specifically Srinivas and Patnaik's approach, is challenging. Their approach relies on a fitness function having an average value that makes sense, as well as a good way of ordering the individuals. This proved challenging to translate, but was achieved by slightly

modifying the fitness function. The new fitness function, f', related to the original as follows:

$$f'(c) = \sqrt{\frac{f(c)}{\sum_{r \in (r_1, r_2 \ldots r_n)} \binom{n}{r}}}$$

The denominator is the number of possible illegal cliques in the graph (the maximum value of f). It is easily seen that the f' orders the individuals in exactly the same way as f. However it was empirically determined that f' fared better than f when using Srinivas and Patnaik's parameter control technique, as the discrete nature of the fitness function was mitigated by the fact that worse fitness values were better grouped together.

One concern, especially important when developing the implementation of Lis and Lis's technique was that while the best solution would always get better and never worse, as the solution converged to a perfect one, and convergence slowed, epochs would have very little effect on the parameters. It was considered that there should be some way to "break ties", in the event that none of the populations were making progress simply because progress was harder to make. Ultimately this was not done, because to do so would be to impose a value judgement on populations that it is hard to justify without strong empirical evidence.

One solution considered, for example, was to evaluate the average fitness of the population in the event that the best individual was tied. But the issue with this was that it unfairly punishes populations with higher rates of mutation (whose average solution quality at any given generation is likely to be lower), without any discernable benefit to the performance of the algorithm. Other solutions, for example involving measuring average solution quality for those solutions that would be brought forward to the next generation, or measuring the diversity of the population (using Hamming distance between the genomes) were rejected for similar reasons, that while it kept the parameter values moving, this movement had no inherent impact or merit.

4.2.3 Debugging Non-Deterministic Approaches

Being non-deterministic by nature, debugging genetic algorithms can be challenging, in terms of both discovering bugs, and the time taken to run experiments to determine whether they have been fixed. Discovering bugs can be challenging with genetic algorithms, as it is both difficult to determine what should and should not be expected behaviour (this is a series of experiments, it is not known beforehand how well they perform when implemented "correctly"), and it takes a significant amount of data to determine whether a bug is actually present in the software.

With respect to the time investment required to solve these problems, often they are the result of representation issues or a difficult translation from the theory to the problem

domain itself. Exactly what the problem is can be challenging to determine, and developing a solution to a semantic problem is often far harder than finding solutions to simpler, syntactic ones.

There was no way to mitigate this problem, apart from ensuring that the codebase was as clean as possible so as to allow for the fast discovery and fixing of bugs. The modular approach certainly helped in this regard, as it was usually easy to identify which module was faulty.

4.2.4 Variable Problem Size

A general "solver" for this problem type needs to be able to accommodate both wild variances in problem size and problem structure. Structurally, most problems are the same – all planned experiments differ only in the number of permissible colours in the colouring (all monochromatic structures to be avoided are complete graphs of various sizes, not $K_n - e$ or other variants), however the variance in size needs to be accounted for.

For manually tuned parameters, this means that they should be re-tuned for each problem instance, a process which takes a lot of time but is necessary in this case. For example, the differing size of the genome makes mutation more impactful on larger problem instances, so the optimal mutation probability is likely to be lower. The values used will be recorded in section A of the Appendix.

For the other techniques under examination, differences between problem instances will not be manually adjusted for to the same extent. The k_n values involved in Srinivas and Patnaik's technique will be adjusted to compensate slightly, but the purpose behind these approaches is that manual adjustment is superfluous to requirements, and that optimal values will be achieved without significant human intervention. Therefore, the available levels and starting levels for Lis and Lis's technique will not be changed between experiments.

5 Experiments and Results

This section will contain an overview of the methodology behind the experiments, followed by graphs of the results of each one. The results are presented without comment and will be further analysed in the next section.

5.1 Methodology Overview

The experiments were run in a shared `tmux` session and left to complete; they took a range of time varying from two hours to approximately a week. It is worth considering how Lis's

technique is to be compared to the techniques that only use one population, as it seems to have a performance advantage, having a population of about 6 times the size of the other techniques (while there are 9 populations, each is smaller on average). As a result, however, it consumes far more resources. The comparison is deliberately based on convergence per generation, rather than per unit time or by memory-time used, so Lis's technique will be measured alongside the others. It is worth noting that an effective population size increase generally does not have a meaningful impact on the convergence of a genetic algorithm, past a slightly better "best individual" in the first few generations.

Each "trial" was left to run for around 500 generations, as this is enough time to form meaningful observations about the convergence rate of a schedule. In order to obtain a reliable average of the convergence, multiple trials were conducted for each schedule. Lis's technique usually received fewer trials than other schedules due to its increased computational cost, usually a third as many as the other trials, which resulted in a similar running time on average. Srinivas and Patnaik's technique ran approximately as fast as the genetic algorithm with static parameters (it has a fairly small overhead for the recalculation of the parameters for each generation), and so the same number of trials were conducted for each of those.

The experiment sizes were chosen such that a range of sizes, diagonal and off-diagonal Ramsey numbers, and experiments with 2, 3, and 4 colours were selected. Most of the attempted colourings are known to be the maximum size possible, with the exception of the $(3,3,3,3)$-colouring of K_{50}, which is one below the current lower bound of $R(3,3,3,3)$, the precise value of which is currently unknown.

For the static approach, the parameters were tuned individually for each experiment. This was performed using independent binary searches on each parameter, relying on the assumption (theoretically justified, and in this case empirically true) that the performance of the genetic algorithm is unimodal over crossover and mutation rates, and that the effect of crossover and mutation rates are independent. This is the standard approach in genetic algorithms to statically tuning parameters. The short experiments to determine the optimal values were not recorded, but the values used for crossover and mutation for each experiment are available in section A of the Appendix. A good population size was determined in a similar way, but it was decided that this would remain invariant over the experiments, as the changes in graph size did not impact that parameter directly.

For Srinivas and Patnaik's technique, their recommended values for the parameters k_1, k_2, k_3, k_4 were used, and a base probability rate of 0.005 was used as advised by the authors in order to maintain a base rate of diversity in the population.

For Lis and Lis's approach, it was decided after some brief experimentation that epochs

	s	p_c	p_m
Initial low value	35	0.17	0.04
Initial medium value	40	0.2	0.05
Initial high value	45	0.23	0.06
Minimum value	15	0.1	0.01
Maximum value	75	0.3	0.09
Value change step size	1	0.01	0.002

Table 1: Table of parameter values for the Lis/Lis schedule.

would occur after every 5 iterations. This is perhaps somewhat on the short side, especially compared to what the authors recommended, but this decision was made based on the experiments conducted in order to properly adjust for the rapid initial convergence that the population experiences. As for initial values for the parameters. Table 1 provides a comprehensive list of the values used, determined after some experimentation. It was decided that these values would be the initial values for all experiments; the purpose of Lis and Lis's schedule is that these values settle into appropriate values for the given problem instance, so there was little point in tuning them prior to the experiment.

5.2 Small Experiments

5.2.1 $(3, 5)$-colouring K_{13}

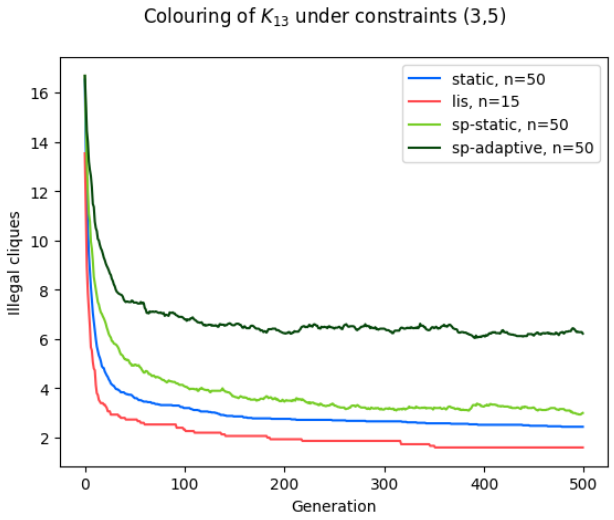

Figure 6: Comparison of schedules for $(3, 5)$-colouring K_{13}

5.2.2 $(4, 4)$-colouring K_{17}

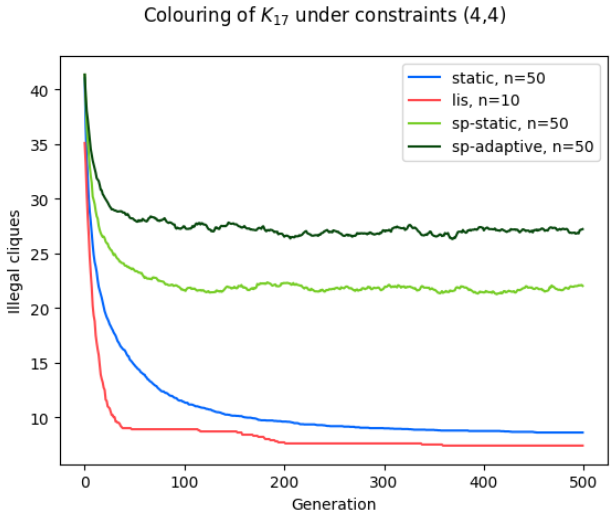

Figure 7: Comparison of schedules for $(4, 4)$-colouring K_{17}

5.3 Large Experiments

5.3.1 $(4, 5)$-colouring K_{24}

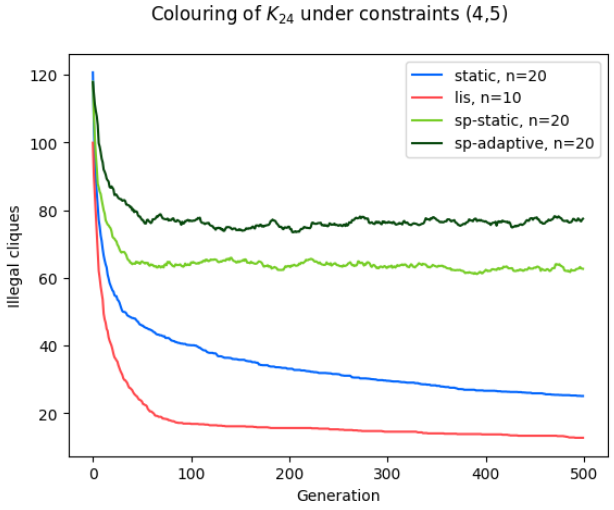

Figure 8: Comparison of schedules for $(4, 5)$-colouring K_{24}

5.4 Multicolour Experiments

5.4.1 $(3, 3, 3)$-colouring K_{16}

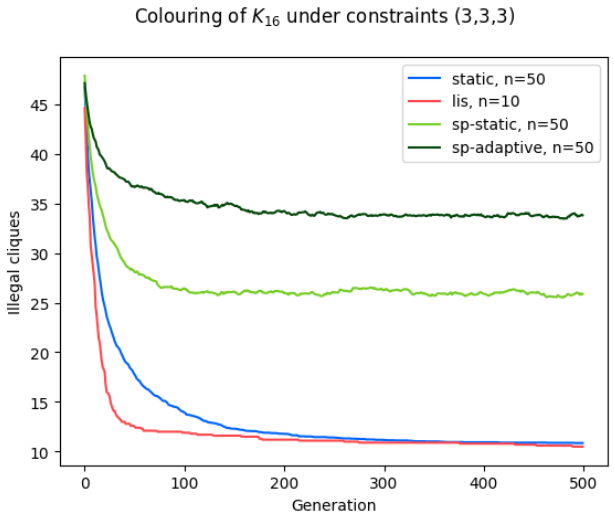

Figure 9: Comparison of schedules for $(3, 3, 3)$-colouring K_{16}

5.4.2 $(3, 3, 4)$-colouring K_{29}

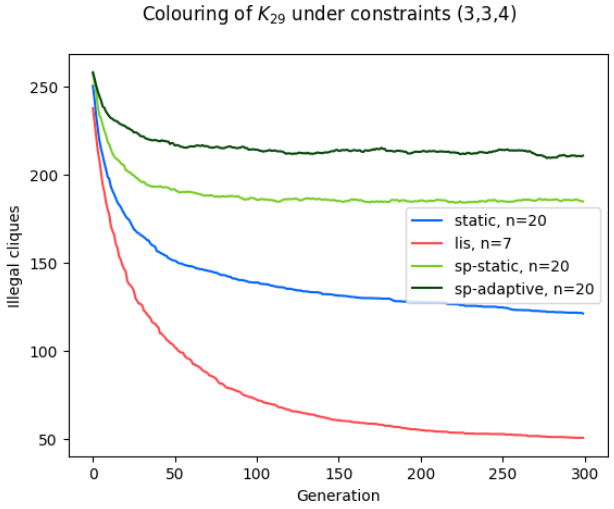

Figure 10: Comparison of schedules for $(3, 3, 4)$-colouring K_{29}

5.4.3 $(3, 3, 3, 3)$-colouring K_{50}

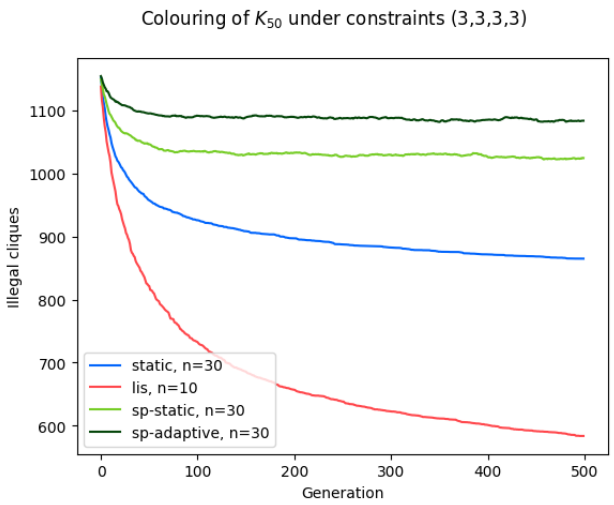

Figure 11: Comparison of schedules for $(3, 3, 3, 3)$-colouring K_{50}

6 Analysis

6.1 Performance of Srinivas and Patnaik's Parameter Control

Srinivas and Patnaik's approach did not prove successful. It is known that selecting poor values of crossover and mutation when tuning parameters for a genetic algorithm can, if done poorly, yield results no better than random search, and it appears that this technique provides an excellent demonstration of this.

In all experiments both variants of the technique performed considerably worse than parameter tuning, and seemed to stop converging completely on larger problems after about 100 generations. There are many reasons behind the total failure of the technique to produce comparable results. While at first sight a plausible explanation for the poor results using this method is a failure of implementation, and that there was some bug that prevented the algorithm from working "properly", this is not the case. Upon detailed examination of log files the crossover and mutation probabilities were as expected given the specific instance at that point – this was confirmed by manual calculation. Therefore the result is not likely to be an implementation failure but rather an accurate assessment of the poor performance of the technique on this particular problem.

6.1.1 Adaptive vs Static Crossover

The technique was split into two different strains: one in which both crossover and mutation probabilities varied with the diversity of the population, and one in which the mutation probability was varied this way but the crossover probability was static (and tuned beforehand). The latter technique performed consistently better than the former, a fact which was noted during testing (and is the reason that both variants were examined in the experiments).

The difference in the performance between the two techniques was fairly constant throughout the set of experiments, which indicates that the negative effect of adaptive mutation does not change with the size or type of the problem instance.

6.1.2 Shortcomings of Approach

The entire point of Srinivas and Patnaik's technique was that a section of the population is used to search the space, in the hopes of maintaining population diversity and ensuring that very different colourings were represented. However, it seems that this local exploration is completely wasted on instances of this problem, as such an exploration is generally no better than random search. This can be deduced from the experiments, in which the best colouring, even after 2-300 generations, contained only 10-25% (depending on experiment size and type) fewer illegal cliques than random. This is because slightly over half of the population, in every generation, is being employed in something approaching a random search. This problem persists from generation to generation and causes real problems for the algorithm, as most individuals created using the worse individuals from the previous population are themselves of a much lower quality than average, meaning they are employed to search the space, and so on.

Even the elements of the population which are not worse than random, but are also not significantly better (those in the 50^{th}-70^{th} percentiles for solution quality) also suffer from increased rates of crossover and mutation when compared to static, tuned parameters, rendering them less useful to the population. The effects of adaptive parameters, in this case, are an approximately 70% reduction in population size, and a much less effective search as a result.

Another factor is the small "lower bound" for the mutation probability, which was used in order to ensure if the population was very homogeneous and had a low variance of fitness, that there would still be some movement in the population. This may actually have had an adverse effect on the convergence, as opposed to being a useful mechanism to overcome convergence to local optima. It was never the case during a run of this technique that the population converged in such a way that this mutation probability was useful, however it did

pose the small risk of mutating a high quality individual of the population into something (in all likelihood) considerably worse. In fact, with the chosen value (which was about 0.005, as suggested by Srinivas and Patnaik). This means approximately 1 in every 200 edges are changed *at minimum, regardless of the solution quality.* On some larger problems, this can be a reliable small change to the graph. Consider the largest experiment for example: K_{50} contains 1225 edges in total. On average, then, each generation will modify 6 edges of all candidate solutions, at random, which for solutions of a high quality is likely to significantly and negatively impact its score. In hindsight, this was an experimental error which should be acknowledged – while most parameters were adjusted taking into consideration the size and nature of the problem, this parameter was not. However, it is unlikely that this had a larger impact than the simple unsuitability of the technique, especially given the poor performance on smaller instances where this was less relevant, although it could be a contributing factor to the slightly worsened relative performance on larger problem instances, on which this problem is exacerbated.

6.2 Performance of Lis and Lis

6.2.1 Difficulty of Comparison

As mentioned in the methodology overview, this technique is hard to compare directly with the others; its structure is fundamentally different to the point where simply comparing it with the approaches can seem unfair. But the direct comparison is actually justified, however – a greater population size does not necessarily mean (and often does not mean) a faster rate of convergence given the relative difference in quality of the starting populations.

Furthermore, while Lis's approach is more computationally expensive, it is also parallelisable on a network of multiple computers. In the existing setup, the different populations are handled concurrently on the same thread – the implementation of the approach did not use multiple threads, partly because the speedup would have been minimal, and partly because the fitness function, the more expensive operation, was using all of the threads anyway, so the speedup versus the other approaches would not have mattered.

6.2.2 Benefits of Approach

As can be seen from the experiment results, Lis's technique exhibits vastly superior performance to both Srinivas and Patnaik's technique, and the technique using static parameters, a superiority that is only more apparent on the larger, more challenging problems. Figure 11 is a particularly striking demonstration of this, with Lis's technique finding a colouring

with only 600 illegal cliques on average after 500 generations, as opposed to the nearly 900 which the static technique obtained.

It is clear that Lis's technique performs at least as well as the static technique on all cases, and performs relatively better the harder the colouring is. Its initially quick convergence should come as no real surprise, as the approach is essentially taking the best of the work of its 9 populations – in this instance the comparison is markedly less fair, as initial performance is bound to be stronger for the best individual out of 9 generations than 1 generation.

This is not the sole reason for the good performance of the approach however, and it seems as though the adaptive effect of parameter control through parallel farming actually has a profound effect, especially on larger problems. One reason for the marked improvement on larger problems is that the epochs are effective. With a step-wise fitness function such as the one being used here, epochs do not actually happen all that often on smaller problems. Consider, for example, the $(3, 3, 3)$-colouring of K_{16}, a problem instance in which its performance was no better than the static genetic algorithm. An epoch occurs every 5 iterations, and when the best individual in the population has few illegal cliques (in this problem instance it was reduced to 12 on average after 100 generations), then the likelihood of the best individual being better than the best individual at the previous epoch is very low, because meaningful progress is usually not made over the course of 5 generations. So the values remain the same and the epoch has proven ineffective. Compare that with the $(3, 3, 3, 3)$-colouring of K_{50}, where every generation usually improves the solution quality by 1, and so every epoch has an effect on the parameters. Consider that this is a limitation of this problem type, specifically the fitness function not allowing for a nuanced modification of the parameter values unless progress has been made, which for smaller problems happens less frequently.

6.2.3 Shortcomings of Approach

The shortcomings of this approach are closely linked to its benefits: once a very good colouring has been found, the adaptive nature of the algorithm will cease to work effectively. This is because epochs will have a lower chance of modifying the crossover and mutation rates, which in turn is due to the fact that convergence will happen at a slower rate. There are ways to mitigate this issue, however. Using Lis and Lis's method to get a colouring that is of a high quality, and then attempting to use some sort of local search from there could work. Alternatively it is possible that the parameters which have been settled on by that point continue to prove effective.

It is worth wondering whether, as indicated in some of the smaller experiments (figures

6, 7), the static genetic algorithm will at some point "catch up" with Lis and Lis's technique. This seems unlikely, as the gradient of convergence in those two examples was never much greater for Lis and Lis's approach than the static approach, and it seems that Lis and Lis's approach never produced any tangible advantage on the smaller problems other than a head start on a good colouring after approximately the first 20 generations. The gradient of Lis and Lis's approach in figure 11 for example is steeper than the static approach even after 500 generations, so it seems unlikely that the two will converge before Lis's approach finds a valid colouring, however long that takes.

This problem is not as bad as it first appears for another reason, that larger problems are the ones that need to be solved in practical applications, and so the problem of not working as effectively on smaller cases is not particularly relevant in practice.

7 Conclusions

7.1 Suitability of Studied Techniques

7.1.1 Srinivas and Patnaik

It can reasonably be concluded from the performance of the technique and Srinivas and Patnaik's approach is not suitable in this problem domain. It has not proven, in any of the experiments, a tendency to produce results significantly better than random, and certainly no results better than an approach using parameter tuning.

It is hard to envisage any method that could conceivably adapt this approach to better fit the problem, it seems to be simply a case of a technique and a problem domain that do not work well together.

7.1.2 Lis and Lis

Lis and Lis's approach is more suitable than both the static method and Srinivas and Patnaik's method; it has shown greater reliability and better performance in the experiments, and overall seems like a good genetic-algorithm based approach to the problem type.

The approach is better in practical settings as well, as mentioned the instances where it performs well, larger problems, are the problem instances that need to be solved and are at the forefront of mathematical knowledge.

It is certainly possible that this technique, in a massively distributed program similar to GIMPS (the Great Internet Mersenne Prime Search, a collaborative project that uses a distributed network of computers to find Mersenne primes) could be effective at producing

real-word results for lower bounds of Ramsey numbers. The technique lends itself to this type of structure very well due to its master-slave architecture, and it could be further distributed by leasing out individual fitness calculations to other computers, much in the same way that the implementation in this project multithreaded the fitness function.

7.2 General Suitability of Genetic Algorithms to Ramsey Theory

This section provides a discussion about the general applicability of genetic algorithms to finding colourings, given the experimental results seen in the previous section.

7.2.1 Discreteness of Fitness Function

The discrete nature of the fitness function did prove to be a problem, especially for Srinivas and Patnaik's technique. It caused issues for Lis and Lis's technique as well, as epochs measure relative progress, and when progress is only discretely quantifiable (and difference in progress is likely to be zero rather than a small decimal) then that becomes a poor measure.

Attempts to change this could be made, although it seems difficult to work out a fitness heuristic that is both admissible and continuous. To do this one would need to examine something other then the illegal cliques in the graph, as there would need to be some way to "split ties". How can it be determined which of two graphs, both containing the same number of illegal cliques, is closer to an optimal solution?

One way to improve the fitness function would be to use a probabilistic colouring model for the edges, allowing for a continuous model of the graph, and a continuous fitness function. This approach was rejected earlier, but given the results and the challenges posed by the consequences of the discrete model, may be worth revisiting. Consider the case of a 2-colouring, and a representation of the graph that instead of 1 meaning red and 0 meaning blue, uses a float value between 0 and 1 to indicate the probability of the edge being red. Consider further the natural fitness function to pair with that, a fitness function that, for each clique, calculates the probability of it being monochromatic and sums this value over all cliques.

This fitness function immediately has some problems, notably an incentive to be unsure. The fitness of a colouring of K_{42} where every edge has a probability of 0.5 to be red is $\frac{\binom{42}{5}}{2^9} = 1661$, and while this isn't a particularly strong fitness value (it is, in fact, the mean value of fitness of a 2-colouring of K_{42}), it is a plateau, which genetic algorithms generally do not fare well with. There is also no easy way, using this fitness function, to extract a solution from a good colouring. Say for example a good solution is found, with a relatively low fitness – how can one extract a colouring from this value? Natural answers include actually

colouring each edge red with the given probability, or rounding the probability to produce a colouring. But neither method is particularly good: the first method actually means that the representation is no longer of a colouring but rather a distribution of possible colourings, which is not particularly useful as a means of producing specific colourings. The second one does actually produce a colouring, but then the problem is that the fitness function has not actually been made continuous, but merely assigns multiple alternate expressions of the same colouring.

The fitness function is discrete because the problem being solved is fundamentally discrete, and that aspect of the problem is a drawback to any potential approach to it involving genetic algorithms.

7.2.2 Multimodality of Search Space

It was mentioned earlier in the paper that genetic algorithms were not especially suited to, and do not perform well on problems which have a very multimodal search space (although they are very good at coping with a relatively small number of local optima), specifically that the necessity of exploring local space essentially at random simply ends up wasting resources, given that the chance of finding good solutions in the proximity of equally good solutions is low.

While adaptive techniques for genetic algorithms can help to alleviate this problem (Srinivas and Patnaik's technique ended up not helping, because it even further utilised the random exploration of local space), the problem is still multimodal and more focused searches would likely perform more effectively.

7.2.3 Suitability of Operators

Crossover and mutation are the most widely-used operators with genetic algorithms, but it is worth considering how effective they proved to be, and to what extent better operators can be devised.

Crossover is a curious operator in the context of this problem, as it tends not to be particularly effective overall. While tuning the parameters for the static technique it was observed that crossover rate did not affect the speed of convergence significantly compared to the rate of mutation. This is because once a solution of good quality is found, and the search is operating in the vicinity of that solution, not much information can actually be exchanged. Fairly early on in the convergence it becomes only a few mutations which are shared between an otherwise completely homogeneous population.

Crossover also has the drawback, in this particular implementation, of not being partic-

ularly meaningful. The genome undergoes crossover, but what does that mean? In practice, this entails (in single-point crossover), the connections involving vertices $k+1$ through n, and some of the connections involving vertex k, from one colouring being combined with the complement of that structure in another colouring. If the crossover was more idiomatically correct (selecting an arbitrary subset of nodes, and swapping all connections where one node is in the chosen subset), the problem of homogeneity reducing the impact would still occur, however, and so it seems unlikely to be much of a problem.

Mutation does work as well as expected, and other, local search based approaches to Ramsey's theorem use a similar method of changing a certain number of edges in order to improve the colouring [2, 18]. In this respect mutation is a good operator and is suitable, however it may be preferable not to modify edges at random, but rather to modify them based on the quality of the colouring, for example only changing edges in an illegal clique, or by some heuristic that is more deterministic.

A different operator entirely could be employed, and some genetic algorithms do this, but the core problem remains, that fundamentally random operators are not particularly effective in the search space, and so the operators available to genetic algorithms do not lend themselves well to this problem.

Based on these factors, either other global search heuristics and techniques, or more local and constructive methods, similar to those employed in [2], are likely to yield better outcomes when attempting to find colourings to advance the lower bounds of Ramsey numbers.

7.3 Author's Assessment of the Project

While this project was narrow in scope, it represents an interesting contribution to the area of research. It provides evidence of the general unsuitability of genetic algorithms to the problem domain of finding valid colourings of graphs, while offering a performance comparison of techniques within genetic algorithms. It is hoped that the conclusion will deter others from attempting to find new lower bounds from Ramsey numbers using the individual-based parameter control described, and encourages either more attempts based in local search techniques or more attempts using distributed systems.

This project was a challenge in several respects: in the extensive research and review of surrounding literature, the practical application of a genetic algorithm in an unusual context, and in the setting, running, and analysis of experiments to compare several vastly different techniques. This project was limited by the hardware available to a certain extent; although finding novel results was not the aim of this project, a simple extension and improvement to this project would be the running of the software on improved hardware for longer periods

of time. As mentioned, the software could also be reworked to run on distributed systems for both a better technique comparison and potentially novel results.

7.4 Further Work

7.4.1 Genetic Algorithms

It is conceivable that other techniques from parameter control that were not examined in this project could prove effective in finding colourings.

One avenue for potential further exploration would be running the algorithms on a distributed system, and investigate other techniques that would benefit from such a comparison. Lis and Lis's approach would work especially well on a distributed system, and performance comparisons in terms of time would be considerably fairer, which is one of the areas of the methodology and experiment design that could have been improved.

Additional testing with alternate approaches is another possibility. In the literature review, additional methods were identified, but it was ultimately determined that they were not to be used. Based on some of the experimental observations made, some of these methods have desirable characteristics and may be worthy of future consideration. For example, the paper which used a fixed schedule in order to decrease the mutation rate over time [4] – it seems from the experiments as though this would actually be a good approach to the problem, as increasingly subtle changes are required to the genomes after the population has converged to a reasonably good value.

7.4.2 Other Search Heuristics

There are other global search techniques that may prove more effective at finding colourings of graphs in Ramsey's theorem. Memetic algorithms (sometimes called Hybrid Evolutionary Algorithms or Cultural Algorithms) are one of the more recent areas of interest in evolutionary computation, inspired by both evolution (similar to genetic algorithms), and the concept of cultural "memes"[19]. The structure is understandably very similar to a genetic algorithm, but with additional "individual learning" procedures interleaved with the population-wise evolutionary steps. These individual cultural learning procedures essentially consist of a brief local search during which the individuals either improve by a predetermined amount, or for a certain amount of time.

The hope, with memetic algorithms, is that an individual learning phase is enough to improve the individual members of a population sufficiently before they re-interact with the other members. Analogously, it is similar to learned experiences over the course of a lifetime.

Memetic algorithms have been applied to several problem domains, from pattern recognition, to manpower scheduling, to feature extraction, with a good deal of success.

Applying memetic algorithms to the problem domain at hand, their potential utility lies in the fact that genetic algorithms, when applied to colouring graphs, sometimes can produce results that are not optimal, but small changes that do not cause problems in other areas of the graph can make them far better. These graphs are essentially a form of unforced error. Consider the colouring shown in INSERT FIGURE. There is one illegal clique in that graph, $1, 2, 3$, and it is really easy to remove from the colouring: simply change the edge $1, 3$ from red to blue, and the colouring is a $(3, 3)$ colouring. The fundamental issue with the class of techniques employed in this project is that this change, as simple as it is, would not necessarily occur soon, or in a determined way. Either one would have to wait for that edge to be the subject of a mutation, or one would have to wait until a crossover occurred under exceptional circumstances. Under a memetic technique, however, this ceases to be a concern. This approach has the further benefit of reintroducing some diversity into the population, as the individual learning phase is likely to affect each individual differently.

One simple cultural learning procedure for this problem domain could consist of a random illegal clique in the graph being selected, and the set of graphs formed by changing the colour of one edge in the clique being considered. The fittest graph in that set would take the place of the individual going into the next population-wide generation. This individual learning phase could even be repeated multiple times to decrease the number of illegal cliques by a considerable amount.

Another alternative technique within global search that is open to examination, and has gained traction in recent times, is the Luus-Jaakola optimisation heuristic. This heuristic operates by maintaining a "box" from which it samples potential solutions. Based on the relative quality of the solution, the box becomes narrower. This has some considerable similarity with simulated annealing, another search technique, but differs in that the factor of change is not strictly decreasing, but sampled from a range of values that is strictly decreasing in size. The Luus-Jaakola heuristic has been shown to outperform genetic algorithms under a number of circumstances [20].

Generally, a random initial value, x is selected, and the "search window" is initialised as the entire space. At each iteration, a prospective value is chosen, by taking the current best value and adding a vector randomly selected from the search window. If the prospective solution is superior, it becomes the current best solution, if it is not then the search window gets narrower. This procedure supports fast "general convergence", while leaving the possibility of nuanced, more local, search open, which is a reason behind its success.

This heuristic is generally used in real-number function optimisation, which is markedly

different from the problem domain of Ramsey's theorem, however it has been successfully applied to the problem of Ramsey's theorem; a paper by J. Mange and A. Dunn successfully implemented the technique to Ramsey's theorem and reproduced 8 of the 10 (at the time) known values for Ramsey numbers [18].

Their approach used the number of edges randomly changed in each iteration as the d value, the value of the relative change between iterations, and this seemed to work well. The authors reported that the value of q, the value responsible for narrowing the search window when the exploratory value did not yield an improvement needed to be unusually high, around 0.9982[18, p.63], and was critical for the performance of the algorithm in the case of Ramsey numbers. This further supports the extremely high multimodality of the space.

It seems likely that the best approach to improving the lower bound of Ramsey numbers rests in the constructive approaches tried by Exoo and others however, which build on colourings of smaller graphs and extend them systematically, or start with a good colouring that is not quite valid and attempts to stochastically improve upon it. These have the immediately clear advantage of a lower amount of necessary computation, as the evaluation of entire populations is not necessary. Additionally, local search is inherently more feasible in this problem domain because it is easy to construct solutions that are significantly better than random using simple techniques, and these solutions are generally local to optimal solutions.

A Static Parameters of Crossover and Mutation

Table 2 shows the values of p_c and p_m used for the static runs in all of the experiments, as it transpired that different values worked well under different eperiment conditions.

Experiment	p_c	p_m
$(3,5)$-colouring K_{13}	0.2	0.05
$(4,4)$-colouring K_{17}	0.2	0.045
$(4,5)$-colouring K_{24}	0.15	0.025
$(3,3,3)$-colouring K_{16}	0.2	0.035
$(3,3,4)$-colouring K_{29}	0.12	0.01
$(3,3,3,3)$-colouring K_{50}	0.15	0.002

Table 2: Table of parameter values for the static schedule.

References

[1] JG Kalbfleisch. Construction of special edge-chromatic graphs. *Canadian Mathematical Bulletin*, 8:575–584, 1965.

[2] Geoffrey Exoo. On some small classical ramsey numbers. *the electronic journal of combinatorics*, 20(1):P68, 2013.

[3] Curtis J Kunkel and P Ng. Ramsey numbers: Improving the bounds of R(5, 5). In *Midwest Instruction and Computing Symposium (MICS)*, 2003.

[4] Thomas Bäck and Martin Schütz. Intelligent mutation rate control in canonical genetic algorithms. In *International Symposium on Methodologies for Intelligent Systems*, pages 158–167. Springer, 1996.

[5] Mandavilli Srinivas and Lalit M Patnaik. Adaptive probabilities of crossover and mutation in genetic algorithms. *IEEE Transactions on Systems, Man, and Cybernetics*, 24(4):656–667, 1994.

[6] J Lis and M Lis. Self-adapting parallel genetic algorithm with the dynamic mutation probability, crossover rate and population size. In *Proceedings of the 1st Polish National Conference on Evolutionary Computation*, pages 324–329. Ocina Wydawnica Politechniki Warszawskiej, 1996.

[7] Stanislaw Radziszowski. Small ramsey numbers. *Electron. J. Combin*, 1(7), 1994.

[8] Geoffrey Exoo. New lower bounds for table III. *constructions*, 7:51, 2000.

[9] Brendan MacKay. Ramsey graphs. Online, accessed at `http://users.cecs.anu.edu.au/~bdm/data/ramsey.html` on 2018-03-26.

[10] Ágoston E Eiben, Robert Hinterding, and Zbigniew Michalewicz. Parameter control in evolutionary algorithms. *IEEE Transactions on evolutionary computation*, 3(2):124–141, 1999.

[11] Kenneth Alan De Jong. *Analysis of the behavior of a class of genetic adaptive systems*. PhD thesis, University of Michigan, 1975.

[12] Hans J Bremermann, M Rogson, and S Salaff. Global properties of evolution processes. *Natural automata and useful simulations*, pages 3–41, 1966.

[13] Brendan D McKay and Stanislaw P Radziszowski. R(4, 5)= 25. *Journal of Graph Theory*, 19(3):309–322, 1995.

[14] Geoffrey Exoo. A lower bound for R(5, 5). *Journal of graph theory*, 13(1):97–98, 1989.

[15] Brendan D McKay and Stanislaw P Radziszowski. Subgraph counting identities and ramsey numbers. *journal of combinatorial theory, Series B*, 69(2):203–204, 1997.

[16] David E Goldberg, Kalyanmoy Deb, and James H Clark. Genetic algorithms, noise, and the sizing of populations. *Urbana*, 51:61801, 1991.

[17] Jaroslaw Arabas, Zbigniew Michalewicz, and Jan Mulawka. GAVaPS-a genetic algorithm with varying population size. In *Evolutionary Computation, 1994. IEEE World Congress on Computational Intelligence., Proceedings of the First IEEE Conference on*, pages 73–78. IEEE, 1994.

[18] Jeremy Mange and Andrew Dunn. Luus-Jaakola optimization procedure for ramsey number lower bounds. *Computer Science*, 10(1):57–68, 2015.

[19] Pablo Moscato et al. On evolution, search, optimization, genetic algorithms and martial arts: Towards memetic algorithms. *Caltech concurrent computation program, C3P Report*, 826:1989, 1989.

[20] Bo Liao and Rein Luus. Comparison of the Luus–Jaakola optimization procedure and the genetic algorithm. *Engineering optimization*, 37(4):381–396, 2005.