

# Final Year Project Report

## Final Submission – Final Report

---

# Genetic Algorithm

Real-World Problem-Solving with Genetic Algorithms: A Practical Exploration  
of Metaheuristic Optimisation

Hameed Roleola

---

A report submitted in part fulfilment of the degree of

**MSci (Hons) in Computer Science**

**Supervisor:** Dr Eduard Eiben



Department of Computer Science  
Royal Holloway, University of London

March 31, 2023

# Declaration

This report has been prepared on the basis of my own work. Where other published and unpublished source materials have been used, these have been acknowledged.

Word Count:

Student Name: Hameed Roleola

Date of Submission:

Signature: Hameed Roleola

# Table of Contents

## Contents

|   |    |
|---|----|
| Table of Figures.....                             | 3  |
| Abstract .....                                    | 4  |
| Chapter 1: Introduction.....                      | 5  |
| 1.1 Aims and Objectives .....                     | 5  |
| 1.2 Genetic Algorithms .....                      | 6  |
| 1.2.1 Constraint Satisfaction Problems: .....     | 8  |
| 1.3 Encoding.....                                 | 11 |
| 1.4 Literature Review .....                       | 15 |
| Chapter 2: Genetics Overview.....                 | 18 |
| 2.1 Genetic Drift .....                           | 18 |
| 2.2 Theory of coalescence.....                    | 19 |
| Chapter 3: Technical Aspect .....                 | 20 |
| 3.1 Technical Decisions .....                     | 20 |
| 3.2 Planning .....                                | 21 |
| 3.3 Software development .....                    | 23 |
| 3.4 Software Engineering .....                    | 24 |
| 3.5 Proof of concept.....                         | 27 |
| Chapter 4: Experimentation and Results .....      | 30 |
| Chapter 5: Final Deliverables and Conclusion..... | 37 |
| 5.1 Reflection of first term .....                | 42 |
| 5.2 Second term aims .....                        | 42 |
| 5.3 Reflection overall .....                      | 43 |
| Chapter 6: Professional Issues.....               | 44 |
| Glossary .....                                    | 46 |
| Bibliography.....                                 | 47 |
| Appendix.....                                     | 49 |

## Table of Figures

|   |    |
|---|----|
| Figure 1: Flowchart of genetic algorithm .....  | 6  |
| Figure 2: Illustration of csp (link).....   | 8  |
| Figure 3: TSP example .....   | 9  |
| Figure 4: Knapsack problem .....  | 10 |
| Figure 5: Algorithms phases for GA.....   | 11 |
| Figure 6: Model of simulated annealing (link) .....   | 15 |
| Figure 7: Point of ref from Data analysis Teaching Week 2: Background .....   | 16 |
| Figure 8: Model of Newton's Method (link).....  | 16 |
| Figure 9: Simple model on GA's (link) .....   | 17 |
| Figure 10: Random genetic drift graphs (link in references).....  | 18 |
| Figure 11: Depicts basic principle behind coalescent (reference) .....  | 19 |
| Figure 12: 2D Prototypes.....   | 26 |
| Figure 13: GA simple program plot .....   | 30 |
| Figure 14: GA TSP program plot .....  | 31 |
| Figure 15: GA Knapsack plot .....   | 32 |
| Figure 16: Simulated annealing Knapsack plot. ....  | 33 |
| Figure 17: Rastrigin Function .....   | 34 |
| Figure 18: A 3D plot of a function subject to a linear inequality constraint. Specifically, it defines an objective function $f(x,y) = x^2 + y^2$ and a constraint $g(x,y) = x + y - 1 \leq 0$ . .... | 35 |
| Figure 19: Visual for TSP GA .....  | 37 |
| Figure 20: Visual for Knapsack GA.....  | 38 |
| Figure 21: GUI.....   | 41 |

# Abstract

*One of the most popular evolutionary computation techniques discovered is Genetic algorithms. A Genetic Algorithm is a type of metaheuristic slightly inspired by the process of natural selection coined by Charles Darwin. Genetic Algorithms are employed to develop high-quality solutions to optimization and search problems. This report sets out to find out how Genetic Algorithms can be used to solve constraint optimisation problems and thus in turn help to solve real-world problems. The main focus of this interim report was on one the constraint optimization problem known as the Travelling Salesman Problem. Through a series of Python programs and visualizations, the effectiveness of Genetic Algorithms in finding high-quality solutions to this problem is demonstrated. This study showcases the potential of Genetic Algorithms to address real-world optimization problems and provides insights into how this technique can be implemented in practical settings.*

# Chapter 1: Introduction

## 1.1 Aims and Objectives

The Genetic algorithm based on Charles Darwin's theory of natural selection, the mechanism that propels biological evolution, is a technique used for solving constrained and unconstrained optimization problems. In more simplistic terms it uses a series of evolutionary processes to solve either with or without given parameters set i.e., constraints. Genetic algorithms contribute to help find solutions to difficult problems and the application of real-world problems, from optimizing travel routes to calculating the survival rate of different organisms.

The aim of this project is to develop a deeper understanding of genetic algorithms and explore their potential applications in solving real-world problems. In particular this project will be focusing on the use of genetic algorithms with constraint optimisation. For the first term I will be working on one problem being the (TSP) Travelling Salesman Problem. The TSP is a classic example of a constraint satisfaction problem for finding the shortest and most efficient route to take within a given list of destinations. This problem is an example of how constraint optimisation has real world applications. Real world applications of this include the optimization of travel, vehicle routing and astronomy to help determine the movement of a telescope for the shortest distance between different stars.

So, the idea I will be working on in my project is '*The use and application of genetic algorithms in constraint optimization problems*'. The project will involve deep knowledge of genetic algorithms, their use in constrained problems, and their practical applications. I chose my project to be in GAs because of their application to real-world problems. Real world problems can vary based on complexity and the exhaustiveness of recourses. An example of this is calculating the survival rate of different organisms, this seemingly tedious task can be handled more efficient with less costs using a genetic algorithm simulation.

For the interim report I decided to begin my focus on one constraint optimisation problem this being the 'Travelling salesman problem' formulated by William Rowan Hamilton. With this I will learning the basics of genetic algorithms and how they can be applied to constraint optimisation. I will be attempting to implement a genetic algorithm for the TSP with research to strengthen my understanding and possibly a visualisation for this problem. I intend to code this using python and other libraries included for example Tkinter and conducting further research to strengthen the understanding of genetic algorithms. The report will also provide insight into genetic algorithms and constraint optimization.

With prior research it can truly be stated the project may prove to be very demanding but with a good use of time and research I believe this will be helpful. To achieve the aim of the project, the dissertation will explore key concepts in the theory of evolution, such as genetic drift, mutations, fitness functions, and crossover functions. The dissertation will also provide a comprehensive background on how evolutionary-driven technology can be used to help solve real-world problems.

In conclusion, genetic algorithms have the potential to revolutionize the way we approach optimization problems in a variety of fields. The Travelling Salesman Problem is just one example of the many real-world problems that can be tackled using genetic algorithms. Through this project, I hope to contribute to the growing body of knowledge in this field and provide insight into the practical applications of genetic algorithms in constraint optimization. While the project may prove to be challenging, I am confident that the skills and knowledge I gain will be invaluable as I continue to pursue a career in the field of optimization and data science.

## 1.2 Genetic Algorithms

**Briefly describe Genetic algorithms and define any terms used in the subject (also put these in the Appendix as a Glossary)**

Charles Darwin's theory of natural selection served as the fundamental foundation for evolutionary algorithms and search heuristics namely the genetic algorithm.

The genetic algorithm (GA), proposed by John Holland in 1975 [4] is a method that utilises natural selection, the mechanism that propels biological evolution, for resolving both constrained and unconstrained optimization problems. A population of unique solutions otherwise known as 'chromosomes' is repeatedly modified by the genetic algorithm. The genetic algorithm chooses members of the present population to serve as parents at each stage and utilises them to produce the offspring that will make up the following generation. The population "evolves" toward the best option over the course of subsequent generations.

The genetic algorithm can be used to tackle several optimization problems, including those where the objective function is discontinuous, nondifferentiable, stochastic, or highly nonlinear and are not well suited for typical optimization algorithms. When some components must only have integer values, mixed integer programming problems can be solved using the evolutionary algorithm.

Three fundamental rules—selection, crossover, and mutation—are used by a genetic algorithm to produce the next generation from the current population:

- **Selection:** chooses the parents, who will contribute to the population of the following generation.
- **Crossover:** combine two parents to create the next generation's offspring.
- **Mutation:** subjects each parent to random modifications.

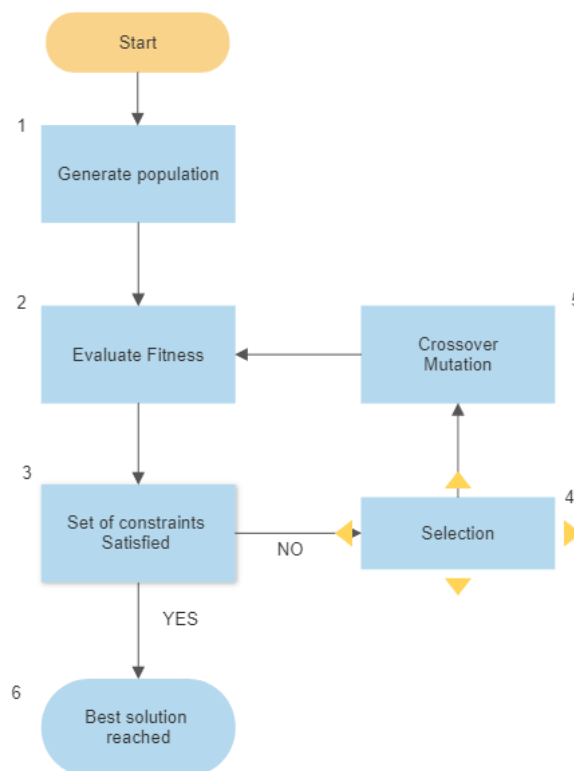


Figure 1: Flowchart of genetic algorithm

Here is an example of a simple Genetic algorithm referenced from '*An Introduction to Genetic Algorithms by Mitchell*' –

1. Start with a randomly generated population
2. Calculate the fitness of each chromosome in the population
3. Repeat the previous steps until an offspring has been created
  - a. Select a pair of chromosomes from the current population
  - b. Cross over the pair at a randomly chosen point to form two offspring
  - c. Mutate the two offspring at each locus
4. Replace the current population with the new population
5. Go to step 2

As this will be important going on I decided to expand on the different types of selection.

In Gas, the selection process involves choosing individuals from the population to be parents for the next generation. There are different types of selection methods, but the two most common ones are roulette wheel selection and tournament selection.

**Roulette Wheel Selection:** Roulette wheel selection is a stochastic selection method where the probability of selecting an individual is proportional to its fitness value. In this method, all individuals in the population are assigned a fitness value based on their fitness function score. The sum of fitness values of all individuals is then calculated, and each individual is given a slice of the roulette wheel that is proportional to its fitness value. A random number is then generated, and the individual whose slice contains the generated number is selected as a parent. This process is repeated until the required number of parents is selected.

**Tournament Selection:** Tournament selection is a non-stochastic selection method where a group of individuals is randomly selected from the population, and the individual with the highest fitness value in the group is selected as a parent. The size of the group is typically a parameter that can be adjusted. This method is repeated until the required number of parents is selected.



### 1.2.1 Constraint Satisfaction Problems:

The main focus of this project regarding genetic algorithms will be constraint satisfaction problems.

Searching can often be executed more easily in cases where the solution rather than corresponding to an optimal path, the only requirement is to satisfy a set of defined conditions. These problems are known as Constraint Satisfaction Problems (CSP). For example, in the n-queen problem (chess-based problem) the set condition is that no two queens in the same column, row or diagonal can attack each other. If much more than this was required, it would then become a more general problem rather than a CSP.

#### Problem definition

A CSP consists of:

- A set of variables  $X = \{x_1, \dots, x_n\}$ ;
- Each variable  $x_i$ , a finite set  $D_i$  of possible values (Domain) and
- Set of constraints restricting values that the variables can take.

The following form can be used to define a constraint satisfaction problem (CSP) in a (finite domain). Find values for the variables that satisfy each constraint given a collection of variables, a finite set of possible values for each variable, and a list of constraints. An example of this occurs in production scheduling. To ensure that each work is finished by the specified deadline, jobs must be processed on machines that can only handle one job at a time. Additional examples follow from the notion that an optimization problem can be stated as a series of CSPs. The solution to a CSP includes consistent and complete assignment. Where a consistent assignment dictates that an assignment does not violate any constraints and a complete assignment is where every variable is assigned.

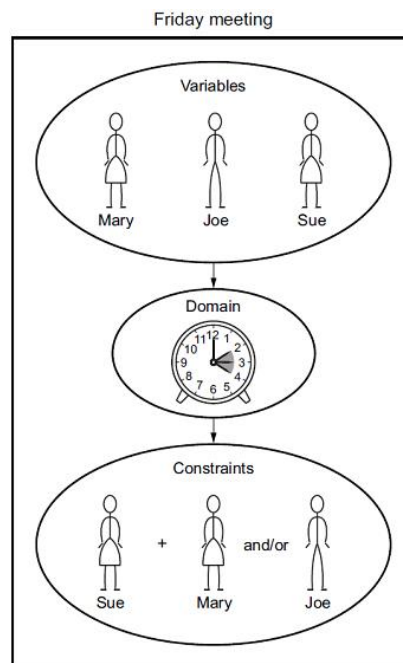


Figure 2 Illustration of how csp works ([link](#))

The following are some CSP that I will explore in this project.

## Travelling Salesman Problem

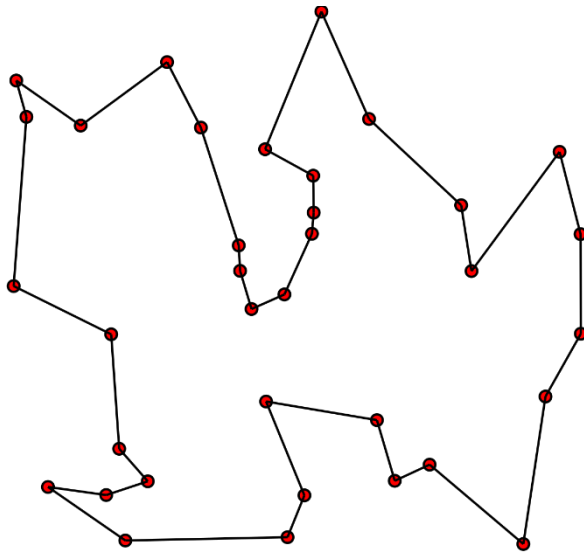


Figure 3: TSP example

[https://en.wikipedia.org/wiki/File:GLPK\\_solution\\_of\\_a\\_travelling\\_salesman\\_problem.svg](https://en.wikipedia.org/wiki/File:GLPK_solution_of_a_travelling_salesman_problem.svg)

### Background and problem formulation

The travelling salesman problem (TSP) is an example of a constraint satisfaction problem which is a classic problem in computer science and optimization. The TSP is described as the task of determining the shortest path that passes through a specified group of cities, returns to the starting city, and visits each city precisely once. It is widely classified as a tough problem as there is no known algorithm that can solve it in a short amount of time. Although this problem may sound simple and possible with the use of brute force method it could be: calculate time to traverse the different possible routes and find the shortest.

However, this is very time consuming in addition to this as the number of cities grows the brute force method becomes infeasible. Real world applications of this include the optimization of travel, vehicle routing and astronomy to help determine the movement on a telescope for the shortest distance between different stars.

For this problem I decided to approach it using a genetic algorithm with a set of constraints. With a genetic algorithm, a population of candidate solutions is iteratively evolved through the application of selection, crossover, and mutation operators, with the aim of improving the quality of the solutions over time.

### Genetic Algorithm solution

A genetic algorithm can be applied to the TSP, a suitable encoding scheme is required to represent candidate solutions. A common approach is to use a permutation-based encoding. This is when each individual in the population represents a permutation of the cities to be visited. The fitness function can then be defined as the total distance travelled by the salesman along the path represented by the permutation.

Different selection operators can be used to select individuals for the crossover, such as elitism, tournament selection or roulette wheel selection. Crossover operators such as order crossover or cycle crossover, can be used to generate new offspring solutions by combining parts of the parent solutions. Mutation operators such as swap mutation or inversion mutation, can be used to introduce new variations into the population.

Overall, the use of a genetic algorithm is a good approach to solving the TSP, which has numerous real-world applications in fields for example logistics, transportation, and astronomy.

### Knapsack problem

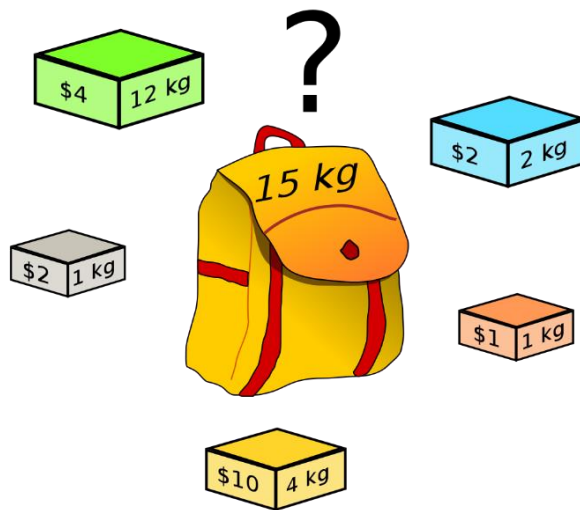


Figure 4: Knapsack problem  
<https://en.wikipedia.org/wiki/File:Knapsack.svg>

### Background and problem formulation

The knapsack problem is a problem in combinatorial optimisation given a set of items, each with a weight and a value, determine the items that could be included in the bag, so the total weight is less than or equal to the given limit and the total value is as large as possible. This problem often occurs in resource allocation which is an example of a real-life problem that can be seen in strategic planning or even economics. Genetic algorithms are shown to be effective in solving optimisation problems and can be implemented to solve the knapsack problem.

The problem can be formulated as a combinatorial optimisation problem as stated earlier where we select a subset of items that maximizes the total value, but with constraints such as the weight of the selected items does not exceed the knapsacks capacity. A binary encoding scheme can be used where each item in the knapsack is encoded as a 1 if it is included in the solution or 0 if not, using the fitness function to evaluate the quality of the solution.

### Genetic Algorithm solution

In order to solve the knapsack problem using a genetic algorithm, binary encoding can be used for the items, each bit in the string represents whether or not the corresponding item is selected. Then we can use a fitness function which evaluates the total value of the selected items. Selection operators can be used such as tournament or roulette, crossover operators such as one point and mutation operators for example bit flip mutation.

To evaluate the performance of the genetic algorithm, different problem instances with differing numbers of items and capacities. We can use different parameter settings such as population size, mutation rate, and crossover rate and measure the performance using metrics such as solution quality and runtime.

### 1.3 Encoding

Genetic algorithms are useful for solving a variety of problems that classified as NP-complete and NP-hard. The overall form of these types of algorithms are described in various places such as “Adaptation in Natural and Artificial Systems” by J Holland referenced below. For the implementation of GA there are important design criteria needed to be considered, one of these is known as encoding:

The encoding determines the representation of a given solution. A gene is any one of the many components that make up an individual. This representation—hence the gene—can be fixed, binary, real, or any other meaningful data structure. The representation used depends on the case.

To build on from the previous point in order to give a good understanding of encoding I will go through the biological background for it. First chromosomes, all living organisms consist of cells which contain a set of chromosomes which are strings of DNA. During reproduction crossover occurs which is when genes from both parents combine to form a new chromosome. This newly created chromosome is then mutated. The fitness of the chromosome is then measured by its survival rate.

There are a few prerequisites problems must have before a genetic algorithm may be used to solve it, these are:

- A method for measuring the quality of the proposed solution i.e., the fitness function in this case.
- Method for representing the solution: numbers, characters.

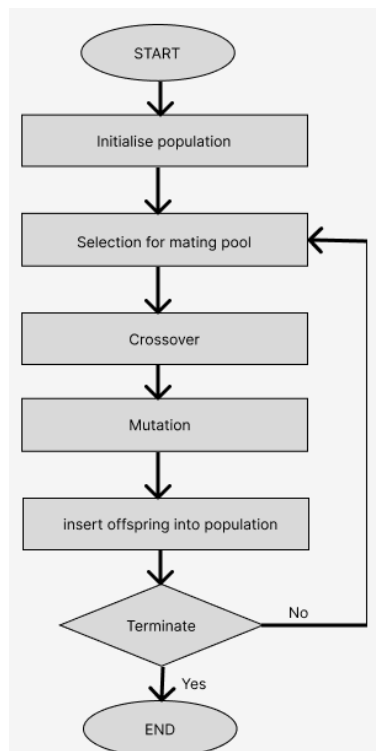


Figure 5: Algorithms phases for GA

Encoding is frequently the most difficult component of using genetic algorithms to solve a problem. The typical approach to express a solution in genetic algorithms is as a string of zeros and ones; this is the most frequent encoding method, as the initial genetic algorithm research employed this style of encoding.

Types of encoding:

- Binary Encoding: An example of a common encoding used for genetic algorithms chromosomes are a string of 1s and 0s, each position in the chromosome then represents a particular characteristic of the problem.
- Permutation Encoding: This type of encoding is typically used for ordering for example the Travelling salesman problem. As stated in the TSP every chromosome is a string of numbers each representing a city to be visited.

Fitness Function

Fitness functions are used to determine how well solutions (such chromosomes) perform. The idea that fitness functions are unique to a particular issue is a key characteristic.

In this case when doing the travelling sales problem, I considered using two fitness functions one from 'Eric Stoltz' and the other from 'YJ Park.'

```
# Use class for fitness function to define inverse of route
# For the fitness function we are trying to minimize the route distance
class FitnessFunc:
    def __init__(self, route):
        self.route = route
        self.distance = 0
        self.fitness = 0.0

    # Distance between 2 cities
    def routeDistance(self):
        if self.distance == 0:
            routDis = 0
            lenRoute = len(self.route)
            for n in range(0, lenRoute):
                fromCity = self.route[n]
                toCity = None
                if n+1 < lenRoute:
                    toCity = self.route[n+1]
                else:
                    toCity = self.route[0]
                routDis += fromCity.distance(toCity)
            self.distance = routDis
        return self.distance

    # Here we calculate the fitness value of the route which is the inverse
    def routeFitness(self):
        if self.fitness == 0:
            self.fitness = 1 / float(self.routeDistance())
        return self.fitness
```

After researching I found that there were already suitable fitness functions created and it would be difficult and a waste of time to implement a completely different one as a result, I decided to use this one from Eric Stoltz [6]. This function makes use of inverse route distance.

```
def total_fitness(total_d):
    if total_d != 0.0:
        #make fitness inverse of total distance
        fitness = 1000000000000000.0 / total_d
    else:
        print("Total distance cannot be zero. Check again")
        sys.exit()
    return fitness
```

Here is an alternative fitness function this fitness function makes use of inverse total distance.

I choose to go with the option from 'Eric Stoltz' as I do not believe the other method makes good use of the actual distance between the routes rather it uses a generic formula. So, the full encoding in this case would not be correct, in the first method we are actually trying to minimize the route distance rather than use a generic formula so we can actually get a proper distance for the routes. From testing I found the initial method to be more accurate than the latter.

### Crossover phase

```
# Crossover of parents to produce a new generation
def crossover_phase(parent_a, parent_b):
    offspring = []
    offspringA = []
    offspringB = []

    # gene from parents set to random values i.e use of random lib
    gene_fromA = int(random.random()* len(parent_a))
    gene_fromB = int(random.random()* len(parent_b))

    # Start and end gene set for sequencing
    Beg_Seq = min(gene_fromA, gene_fromB)
    End_Seq = max(gene_fromA, gene_fromB)

    for i in range(Beg_Seq, End_Seq):
        offspringA.append(parent_a[i])

    offspringB = [item for item in parent_b if item not in offspringA]
    # getting both genes from parents
    offspring = offspringA + offspringB
    return offspring
```

With the mating pool created before, the creation of the next generation can then proceed this is called the crossover phase. In the case that our individuals were strings of 1s and 0s and the rules established did not apply. Then a crossover point [6] could be picked and splice the two strings together to produce an offspring.

The TSP is special in that we must include every location precisely once. We can employ a unique breeding function called ordered crossover to conform to this criterion. Through ordered crossover, we first arbitrarily choose a subset of the first parent string and then populate the remaining route with the genes from the second parent in the order in which they appear, without repeating any genes in the chosen subset from the first parent.

### Mutation phase

```
# will have a set mutation rate, route1 is retrieved from route generated
def mutation_phase(route1, mutate_rate):
    # Set mutation rate outside, cannot set inside loop
    lenRoute = len(route1) # size route

    # swapping around for mutations with random element
    for swap in range(lenRoute):
        if (mutate_rate > random.random()):
            swappedInd = int(random.random() * lenRoute)
            # Had to replace was getting out of index error
            # swappedInd = int(random.random() * lenRoute)

            # Swapping around
            swapped1 = route1[swap]
            swapped2 = route1[swappedInd]

            route1[swap] = swapped2
            route1[swappedInd] = swapped1
    return route1
```

In GA, mutation plays a crucial role in preventing local convergence by introducing new routes that let us explore different regions of the solution space. Mutation would simply entail estimating the likelihood that a gene would change from 0 to 1 if our chromosome included just 0s and 1s. In this case however, to follow the rules we cannot drop cities rather use a “swap mutation”. This indicates that two cities may swap positions along our route with a very low chance.

```
def crossover():
    global population
    updatedPop = []
    updatedPop.extend(population[: int(pop_size*nxt_gen)])

    for n in range(pop_size - len(updatedPop)):
        index1 = random.randint(0, len(updatedPop) - 1)
        index2 = random.randint(0, len(updatedPop) - 1)
        while index1 == index2:
            index2 = random.randint(0, len(updatedPop) - 1)

        # gene from parents set to random index used from above i.e use of random lib
        parent_a = updatedPop[index1]
        parent_b = updatedPop[index2]
        p = random.randint(0, totalCities - 1)
        Offspring = FitnessFunc()
        Offspring.route = parent_a.route[:p]

        notInOffspring = [item for item in parent_b.route if not item in Offspring.route]
        Offspring.route.extend(notInOffspring)
        updatedPop.append(Offspring)
    population = updatedPop
    return
```

As you can see the crossover in the visualisation uses similar principles from the genetic algorithm I previously looked into. We can see how the crossover function creates a new population from the top of a previous generation, replacing any non-essential individuals with the new ones.

Thus, we can conclude that when there is not enough information available to solve a problem, GA can be implemented. To achieve the optimal performance using GA, the problem must be represented with the proper encoding, the appropriate fitness and crossover functions must be determined, and the parameter values must be appropriate.

## 1.4 Literature Review

Survey the relevant literature on GA and optimization methods, including their history, main concepts, principles, and applications. Identify the strengths and weaknesses of GAs and other optimization methods and explain why GAs are suitable for the selected optimization problems.

**The report will include a survey of optimisation methods, including GAs.**

Optimisation is the problem of finding the best solution from all the feasible solutions. It is clear from the opening passage from the book “Numerical Optimization” by Nocedal and Wright that optimisation is essential and used in all facets of life. For example, when designing and operating their production processes, manufacturers aim for optimum efficiency.

Optimisation methods are known as techniques used to find the best possible solution to a problem, often involving finding a specified value of a function either being the maximum or minimum. Example of optimisation methods include genetic algorithms, simulated annealing, gradient decent and Newton’s method. Below is a survey of optimisation methods:

### Simulated Annealing

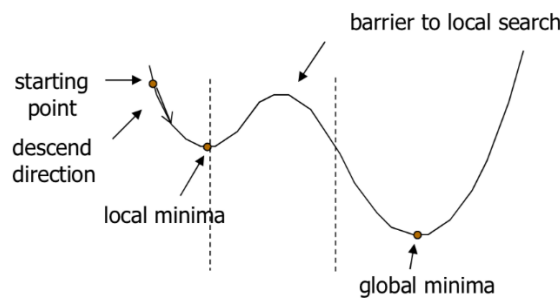


Figure 6: Model of simulated annealing ([link](#))

Simulated annealing [13] is based on the annealing process in metallurgy (extracting metals from their ores and modifying the metals for use). It works by iteratively improving the current solution by exploring the search space in a probabilistic manner. The algorithm begins with an initial solution, then it iteratively modifies this current solution by applying a particular move following this it either accepts or rejects the new solution based on the probabilistic criteria.

An example of an application discussed in [13] is the TSP, simulated annealing applied to find the best approximate solutions. In this application, we are trying to find the most efficient route that covers all cities, and the measure of efficiency is the total distance travelled. Simulated annealing is used which involves repeatedly changing the current route by swapping the positions of two cities. The decision to accept or reject the new route is based on a probability calculation.

Regarding strengths of simulated annealing, the book states that it is a general-purpose optimisation algorithm that can handle a wide range of optimisation problems ranging from non-convex to discontinuous cost functions. The simulated annealing is also quite easy to implement and can be applied to problems with a large number of variables. The weaknesses of simulated annealing include a need to carefully choose the cooling schedule and the acceptance probability function this is to ensure good performance. Simulated annealing can be computationally expensive as well, especially when used on search spaces with high dimensions.



## Gradient Decent

### Gradient descent (GD)

GD algorithms are **iterative** procedures to *locally minimize* a scalar multivariate function, e.g.  $f: \mathbb{R}^d \rightarrow \mathbb{R}$ .

You start from an *initial guess*,  $z^{(0)}$ , e.g. a **random Gaussian  $d$ -dimensional vector**, and update it in the direction of  $-\nabla f$ .

At iteration  $t \in \{0, \dots, T\}$ , you evaluate  $\nabla f(z^{(t)})$  and let

$$z^{(t+1)} = z^{(t)} - \eta \nabla f(z^{(t)})$$

The **number of epochs**,  $T$ , and the **learning rate**,  $\eta > 0$ , are the hyper-parameters of the algorithm.

Figure 7: Point of ref from Data analysis Teaching Week 2: Background

Gradient decent [14] is defined as an optimization algorithm that updates the model's parameters iteratively to minimize the cost function. The update rule can be written as follows:

$$\theta^{(k+1)} = \theta^{(k)} - \eta \nabla J(\theta^{(k)})$$

where  $\theta^{(k)}$  is the parameter vector at iteration  $k$ ,  $\eta$  is the learning rate (i.e., step size),  $\nabla J(\theta^{(k)})$  is the gradient of the cost function  $J$  evaluated at  $\theta^{(k)}$ , and  $\theta^{(k+1)}$  is the updated parameter vector for the next iteration. The goal of gradient descent is to find the set of parameters that minimize the cost function  $J$ .

Applications of the gradient decent includes that it can be used to solve unconstrained optimisation problems, in cases where the objective function is to be minimized over the entire space of the parameter. Its strengths include that it is simple and easy algorithm to implement that can be applied to a wide range of optimisation problems. Furthermore, it can be modified with momentum or adaptive step size to improve the convergence speed and stability. Whereas the weaknesses include that it can converge slowly or not at all if the function is highly non-convex or has flat regions.

## Newton's Method

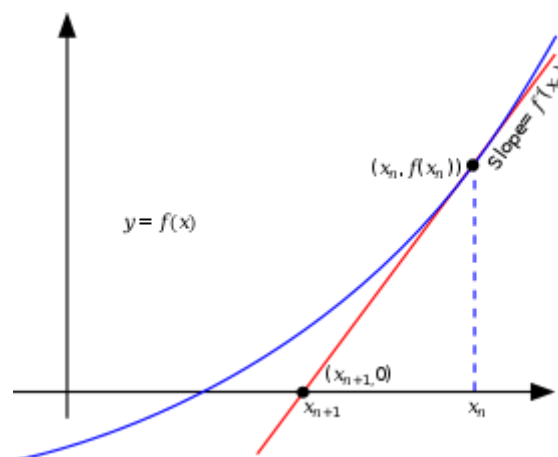


Figure 8: Model of Newton's Method ([link](#))

According to Nocedal and Wright (2006)[15], Newton's method is a numerical algorithm used to find the roots of a function. The algorithm involves generating a sequence of approximations, where each new approximation is obtained by using the tangent line of the function at the current approximation. This results to an iterative process that gets progressively closer to the root. The

method is well-known for its fast convergence rate, especially near a simple root, although its performance could be slower near multiple roots or singular points of the function.

Applications of Newton's method covered from the book are as follows, finding the roots of nonlinear equations, solving systems of nonlinear equations, and minimizing/maximizing nonlinear functions. The author also describes advantages of Newton's method such as its fast convergence rate near a simple root and the ability to handle large-scale problems. However, there are several limitations and weaknesses of Newton's method. For instance, the algorithm might converge slowly or not at all if the initial guess is far from the root or if the function has multiple roots or singular points.

## GA

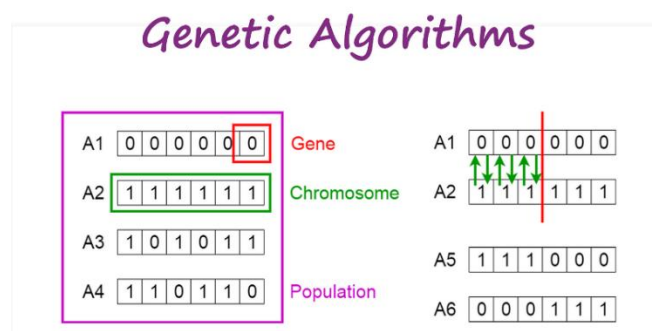


Figure 9: Simple model on GA's ([link](#))

As discussed earlier in the report the genetic algorithm is an optimisation method that uses natural selection as the basis to solve complex problems. This involves creating a population of unique solutions, known as chromosomes modifying them through selection, crossover, and mutation to produce the next generation. The process continues until the population evolves towards the best option over successive generations. The genetic algorithm is useful for solving optimisation problems that are discontinuous and highly nonlinear or require integer values for some components. Overall, it is a powerful tool that has been influenced by Charles Darwin's theory of natural selection.

The application of GA's include that they can be used to solve a wide range of optimization problems, such as engineering design, scheduling, financial portfolio optimization, and logistics. One of the main examples used is the TSP which is discussed earlier.

The strength of the GA is that they can handle non-linear or non-differentiable problems that are otherwise difficult for traditional optimisation methods. Another strength is that GAs is flexible and can be adapted to different problem domains by modifying the selection, crossover, and mutation operators. The weaknesses of a GA include that they can be computationally expensive even more so when dealing with large populations or complex fitness functions. The quality of the solution may differ based on the parameters and operators utilised, and GAs are not guaranteed to find one within a certain amount of time.

Compared to other optimisation methods like simulated annealing and gradient descent, the GA's unique advantage is that they use a population-based approach for the solution. Instead of simply finding one solution at a time, GA's explore many different possible solutions at once. This can help GAs find the best possible solution, rather than regressing with just one good solution. Furthermore adaptable, GAs can be made to work for various types of problems. This is so that different components, such as the operators for selection, crossover, and mutation, can be modified.

## Chapter 2: Genetics Overview

### 2.1 Genetic Drift

Genetic drift can be described as the random variation in the frequency of an existing gene variant in a population.

There are two major types that can occur which are: the founder effect and bottlenecks. The founder effect occurs when a small subset of a larger population establishes a new population. A population bottleneck arises when the size of a population is drastically reduced, causing a shift in the distribution of alleles.

First, we will discuss what causes genetic drift, smaller populations of organisms are significantly more likely to experience genetic drift. As seen on the graphs the lines represent the frequency of alleles in a population. From the initial graph we can see that when the population is small and a lot of alleles exist, each of the alleles can easily go extinct or become fixed in the population. By contrast when the population is large an allele is less probable to disappear entirely since more individuals contain it, reducing the likelihood that all of them would become extinct (seen in the last graph).

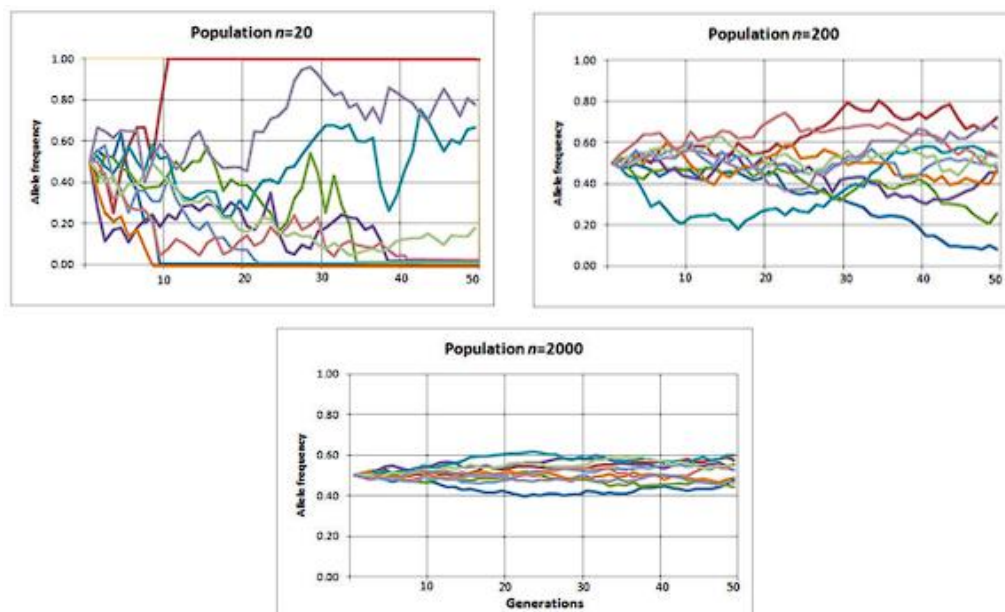


Figure 10: Random genetic drift graphs ([link in references](#))

#### Population Bottleneck

The Bottleneck effect is when you have some sort of major disaster or effect that in turn kills off a lot of the population resulting in a severe decrease in a population's size. Examples of these are competition and spread of disease which leads to a massive decrease in population size. So only a little bit of the population is able to survive. The organisms that did not die now determine the allele pool. Some alleles become more frequent simply because they are the only ones left. An example of this can often not be discussed but is seen in [7] with the case of antibiotics. Antibiotics kill pathogenic bacteria in your system regardless of their alleles and massively help to reduce harmful bacteria, preventing the spread of disease. However, if the patient is not consistent with usage and quits too early then a small population of antibiotics will survive. Potentially, the allele frequencies

in this much smaller population of bacteria could be significantly dissimilar to those in the original population. Selection or further genetic drift causes the allele frequencies to change, and the new alleles will be dominant in the population.

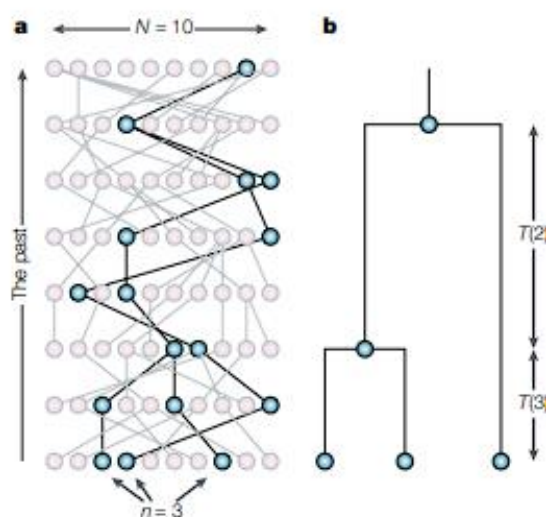
### Founder Effect

The same idea of a population becoming small, but the founder effect isn't a result of a natural disaster rather a new population is formed in a new location. If this new population does not interact with and reproduce with the main population, the allele frequencies in this population will indeed be substantially different from those in the parent population leaving variations being disproportionately different. A result of the founder effect, many islands have species that only exist on a single island. For example [7], perhaps if two birds of a species land on an island, the diversity there will be reflected for simply by their alleles. Although these alleles may initially dominate, population mutations will eventually result in new adaptations. This new adaptation remains in the founding population. The two groups potentially become so different over time that they are unable to interbreed. This is a common way that species diverge.

## 2.2 Theory of coalescence

It represents how coalescent events on a gene genealogy are distributed.

The coalescence theory states that all alleles of a gene must have descended from a single allele. In a population of constant size with random mating, the idea often applies to neutral or relatively neutral genes. The generation period and population size affect the coalescence time, which is the distance from the most recent common ancestor.



This figure depicts the complete genealogy of a population of “ten haploids” taken from [5]. As shown on the figure the lines trace the ancestries of the three sampled lineages to a single common ancestor. In other terms all of the alleles in the final generation go back and coalesce which means merged together in this case merged together in the past. So, we can see the coalescent even or the single allele that gave rise to all of the descendant alleles. We can see that there is no natural selection here so in a way coalescence is inevitable. It should also be noted that the average time to coalesce is dependent on the size of the population.

Figure 11: Depicts basic principle behind coalescent (reference)

## Chapter 3: Technical Aspect

### 3.1 Technical Decisions

Justify your choices for the software used to create prototypes.

One of the main software used for my project was sublime text which was used for coding both the graphical user interface and the genetic algorithm program. I chose to use sublime text due to its simplicity and convenience. Prior to this I had not been aware of sublime text and had not used it, but upon research realised how convenient it is to use. You can work with folders already created and it works with multiple programming languages incorporating different build systems. As you probably know already my program uses different python modules, with pip install and sublime text this was easy to do so using modules like Tkinter was not a problem to implement. There are also many useful keyboard shortcuts [10], again emphasising the convenience such as 'cntrl /' for commenting out code and 'cntrl [' for indenting large chunks of code.

Prior to this as I mentioned before I had not heard of sublime text, so I had other text editors in mind to use such as the python idle, eclipse and visual studio. Although I may use the python Idle in the future, I decided to use sublime text over the other text editors as it was more straightforward to use and as I previously stated its convenience. It has a user-friendly environment and is very intuitive to use, its straightforward to use and it is not difficult to learn how to use. It also has customizable syntax-specific colour schemes are used to display code. The only caveat to using Sublime text is if you are on the free version an annoying pop-up frequently appears.

Other technologies used include pygame and Tkinter, although more were used these the main ones. At first, I decided to try and code the entirety of the project in Tkinter. However, I quickly learned how hard this would be to implement after doing research I found that the majority of people said it would take months to code. As a result, I decided to use Tkinter for the GUI and instead use pygame for the simulation of the TSP. I decided to use pygame since it is the most logical choice for making objects on the screen move and interact with one another. Since Pygame is used primarily for modelling and game development, I thought it would be a decent option.

After using Sublime Text for a while, I realized that it didn't have advanced features that I needed, so at the beginning of the second term, I decided to switch to Visual Studio Code.

One of the main advantages of Visual Studio Code is its comprehensive debugging. This supports multiple languages and provides a range of debugging features, such as breakpoints, variable inspection, and call stacks. These features make it easy to identify and resolve issues in the code, saving me a significant amount of time and effort. Furthermore, the editor provides instant feedback on syntax errors, code formatting, and other issues, enabling me to fix these problems before they become worse. This real-time feedback helps to promote cleaner and more efficient coding practices, making the code easier to maintain.

Overall, while Sublime Text is a useful and convenient editor, Visual Studio Code offers several advantages that make it a better choice for me. Its extensive debugging capabilities and real-time feedback make it a powerful tool, allowing me to write efficient, maintainable code with ease.

## 3.2 Planning

### Describing the beginning phases.

At the beginning of term, we were tasked with creating a project plan which is essentially the guideline and framework for the project outlining what you hope to achieve on a weekly basis. My project plan consisted of doing a lot of research and learning how to use the software I selected for my project namely sublime text. Thankfully learning to use sublime text was not too difficult so I didn't have to dedicate a huge amount of time for it. However, due to the difficulty of the project I and to dedicate a large portion of time to research and understand it. Which justifies why there was some delays in the earlier weeks for the development of the project. With this research time I managed to discover a bevy of research articles and some books on genetic algorithms. Thus, explaining how I was able to find many points for reference.

Naturally, there were days when I was unable to follow my project plan, which caused me to slip behind a little bit, but this was always anticipated because the project plan represented my best-case for what I could accomplish rather than more of a standard plan. As a result, anything less was primarily a great plus for me.

The standard for the online project diary differs slightly from the project plan in that it effectively served as a log of my activities, detailing what I accomplished each week. For the sake of understanding, the project plan is the **framework** for the project outlining what you hope to achieve on a weekly basis, whilst the project diary is a **log** of my weekly activities. As it was clearly emphasised to be important from the beginning, I made sure to dedicate time to it. So, despite not fully following the plan I made sure to update the diary weekly with my activities and what may have been a limiting factor such as me catching covid or using certain technologies which require too much time. I didn't report on a weekly basis because I frequently missed submitting a review of my week and my accomplishments during some weeks due to the heavy workload I was carrying out. At times the last thing I remembered was, of course, to make a report about my week's work because finishing big amounts of work every week was exhausting.

During my meetings with the supervisor, I always had my trusted tablet with me to take notes and write down feedback on certain areas. This came in handy when I was working on the fitness function and didn't know where to begin but as I had my notes from meetings, I was able to go back and review my notes. From this I was able to thoroughly research into fitness functions and discover different fitness functions that could be useful and outline them. Thus, demonstrating the importance of taking notes during meetings.

## Online Project Diary

The full project diary can be found in the appendix.

I've been keeping track of project diary entries since the beginning of the term. Initially I did this by noting down what I was planning to do on my tablet. Then I found it would be a much better idea to create a digital diary on my GitLab repo and update it every week, I felt it would be best to update the diary in weekly blocks. It is stated that we should spend up to 20 hours each week on the project. Initially, I envisioned working 4 hours per day for 5 days, but I quickly realised that this was not realistic due in part to other modules, such as data analysis, that required time as well. As a result, I decided to change this to roughly 3.5 hours per 6 days. Working on the project would entail doing programming, writing a report, and researching concepts. Between the task I aim to spend an hour on each at time alternating at times to guarantee that certain aspects are up to standard. However, I will admit that often maintain this was difficult due other modules but also the workload. This explains why my project diary entries are well organised at the start of the term, but by the middle of November it was difficult to maintain track of this diary, not because my plan was not accurate, but rather the workload was becoming increasingly difficult. You'll notice that I have consistently aimed to give my project diary the significant attention throughout the term and have never attempted to neglect it. Mainly since this was a relatively new feature for me, it was difficult for me to keep track of it constantly. But since this was a new teachable moment, I want to build on it and improve my organisational abilities in order to have more time to devote to the project diary next term. Aside from this, the project diary itself is highly informative because I always mentioned the work I did and the results I got at the time, as well as what went wrong with the project and what I intend to accomplish next.

This shows the importance of the project diary in several ways. Firstly, it demonstrates that keeping track of project progress is a key component of effective project management. By keeping a diary, I was able to plan work and monitor the process in comparison to the diary over time. Additionally, the diary provided a record of my thought processes, allowing me to reflect on what I had learned and to identify areas where I could improve. It also highlights the need for flexibility, initially I set out a detailed plan for I would allocate time, but then quickly realised that this plan was not realistic given other commitments. By adjusting my approach, I was able to find a somewhat of a workable balance that allowed me to make some progress on the project while still meeting their other obligations. This also demonstrates the challenges involved in maintaining a project diary. Maintaining the diary became increasingly difficult as the workload increased, however this a valuable teachable moment that I can build on in the future to recognise the importance of planning. Overall, this shows that a project diary is quite an essential tool for managing project work, providing a record of progress, reflection, and learning. This might be useful in the future as by maintaining a diary, project managers can improve their planning and decision-making abilities, while also building a valuable record of their work that can be used to inform future projects.

### 3.3 Software development

Discuss tools used i.e., Gitlab

From the beginning of term, I have been using a tool called Gitlab with access provided by the university, a platform for version control. The University provided me with a private repository where I had to commit all my code and I decided to commit my weekly logs there too. This is for ease of use and access. This includes the program, report, and project diary.

Using Gitlab allowed me to be able to have a place to consistently upload different components of my project whilst also ensure that if anything is deleted, I still have a backup. With the aid of version control I was also able to go back and look at previous drafts and be informed on what I need to change. This was a good way to present my work as evidence of what I had been doing. It is also safe for me since, in the unlikely event that my IDE crashes and all of my work is lost, I still have a backup option to restore all of my code. I was able to track any modifications I made to earlier versions of my code thanks to Gitlab. I would make sure to commit code or work at least at the end of every week often on separate branches. Committing code enabled me to keep track of the adjustments I made to branches. In addition, I was able to commit with a log message of my choosing. Log messages assisted me in stating what was modified or added to the commit I made, effectively clarifying the commit for future reference if an observer or myself went back to read it. At times I struggled to commit very often as I was too into the work and wanted it to be of more quality, the idea of submitting incomplete or lingering code didn't sit right. So, I did endeavour to try and commit more often

During my project I also implemented test driven development, which is part of the agile methodology. Using this I also implemented unit test to check that the function runs the way it is supposed to. This was immensely helpful because it enabled me to write efficient code while ensuring that the test requirements were well specified. Furthermore, the agile methodology pushed me to conduct a lot of refactoring in and around my code. For the visualisation I also did a series of testing to see how different object would appear on the screen. This was necessary to see the different types of adjustments I would have to make for the visualisation to look accurate and actually encapsulate the travelling salesman problem. Through this I managed to make the necessary adjustments to the code to make sure that the objects appear properly. For example, changing the use of single numbers and appending to just creating a random list. The latter would have worked but would have made displaying the objects very hard to do and through testing creating a list is much easier.



### 3.4 Software Engineering

I used an agile methodology while working on this project. This method performed best for my project since it enabled me to deliver what I had projected based on my project plan. The agile methodology operates traditionally by organising a project and breaking it down into manageable segments. At every stage, you process and plan in addition to continually testing and improving. For me, I had a thorough layout plan (project plan) from the beginning, so I knew exactly how I wanted my prototype to appear.

By using agile methodology, I organized the project into manageable tasks for myself to do over a two-week period following each meeting, which helped me become more focused. Depending on how quickly I completed the minor tasks, this eventually piled up to a much greater project. This promoted greater transparency and increased productivity.

During the development of my project, I had meetings with my supervisor over 2-week intervals. In other words, every 2 weeks on a Monday I have a meeting with my supervisor, and we discussed my current progress and what I have completed up until then. By regularly having these meeting I was a to improve my work at a much better rate and truly break down different problems I was having. Eventually, I was able to exhibit more and more work after each meeting, including a layout concept that was more intricate and different from what I had originally imagined. Demonstrating the effectiveness of reviewing work and having input from an observer as well as how overtime the idea of my project has changed. As a result, the meetings enabled me to exchange useful information and ensure that I always achieved my weekly target. It's not surprising that over time, my project's progress has drifted greatly from how I had anticipated it would in my project plan given the ongoing changes and updates to the plan. Not only this, but my prototype, which I created and was intended to be a quick review of what I anticipated my products to look like, was very different from what they actually looked like.

The key areas where the agile approach significantly benefited me were in enhancing the quality of my product and structuring my report. By applying the agile methodology to plan and organize my report's layout, I was able to progressively see improvements in its structure and overall quality. This approach proved advantageous in both product development and report writing.

#### Design patterns

Design patterns are characterised by their use they include creational, structural, behaviour and architectural.

**Object Scope:** Handles object relationships that are changed in real time.

**Class Scope:** Addresses relationships between classes that can be modified during compilation.

I decided not using the singleton pattern after research I found that this design pattern would not be a suitable chose because it instantiates a class to singular instance. With there being multiple instances in my program I elected not to use this. It is also worthy to note that the standard design patterns are not commonly used in python. In my project I have python code for modelling purely logic, then I have the view i.e., the visualisation and the GUI and the GUI also tried to implement controller. The complexity of the genetic algorithm is hidden and is not shown in the GUI program.

The design pattern that I implemented in the project was the Model-View-Controller (MVC) pattern. This pattern is widely used in software development for building user interfaces and is known for its ability to separate the presentation logic from the application logic.

In the MVC pattern, the application is divided into three main components:

1. **Model:** This component represents the underlying data and logic of the application. It contains the business logic, data access logic, and other components that manage the application state.
2. **View:** This component represents the user interface of the application. It is responsible for displaying the data to the user and handling user input.
3. **Controller:** This component acts as the intermediary between the Model and the View. It handles user input, updates the Model based on user actions, and updates the View to reflect changes in the Model.

In my project I have python code for modelling purely logic, then I have the view i.e., the visualisation and the GUI and the GUI also tried to implement controller. The complexity of the genetic algorithm is hidden and is not shown in the GUI program. The key benefit of using the MVC pattern is that it promotes the separation of concerns, making it easier to maintain and update the application. It also enables developers to work on different components of the application simultaneously, which can speed up the development process.

### **Object orientated programming**

For my project I decided to implement object orientated programming (OOP) as I am familiar with it. OOP is a programming paradigm that organizes code into objects that encapsulate data and behaviour. It provides a way to model real-world entities and concepts in code, making it easier to manage and manipulate complex data structures. A key benefit of it is that it promotes code reusability and modularity. Meaning with OOP, code can be organized into classes and objects that can be reused in different parts of the project, reducing duplication of code, and making it easier to maintain the codebase over time. In the context of the project specifically CSP's, OOP was used to encapsulate data and behaviour related to specific entities and concepts in the project. For example, a class represents an individual or population, with properties such as chromosomes (solutions), as well as methods for handling the processes of a genetic algorithm. Overall, the decision to use OOP in the project was advantageous, as it provided a way to organize and manage the codebase effectively and promoted modularity and reusability.

## 2D Prototypes

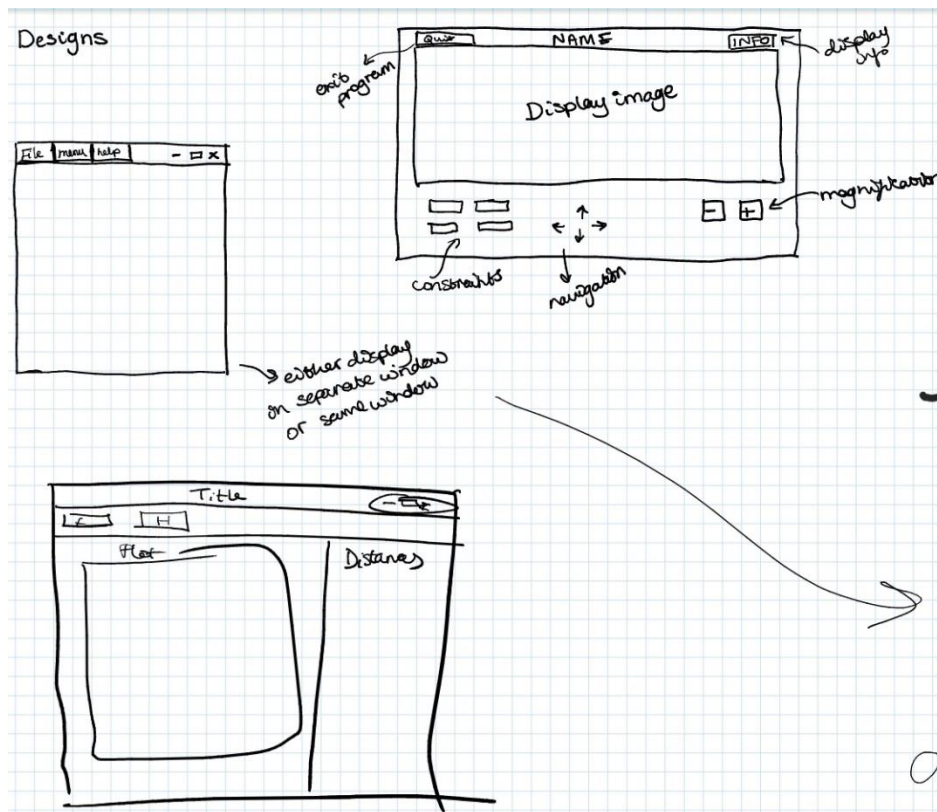


Figure 12: 2D Prototypes

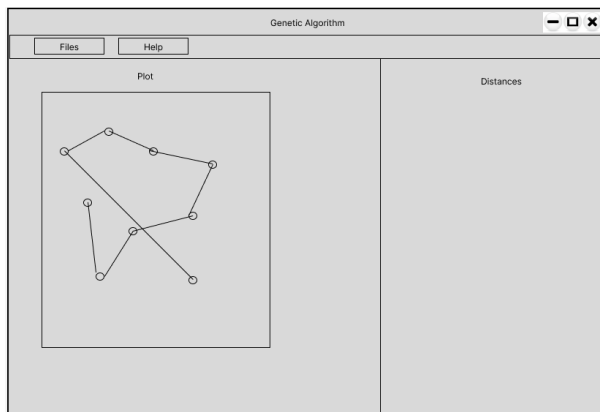
The screenshots you see here are 2D prototypes of work I did earlier in the term, I expected my application to look like these prototypes. As you can see my 2D prototypes differ from my prototype developed. It is important to show how the project developed. This demonstrates how much I learnt from working on the project and how far I have developed. The prototypes are purposefully quite vague in order to give me more freedom when building the application. The key area where the agile methodology most significantly helped me was with raising the calibre of my productivity. Even in my report, I applied an agile methodology. In designing and outlining a thorough plan for how I wanted my report to look, I gradually began to recognise how the quality and format of my report helped me.

### 3.5 Proof of concept

#### First term

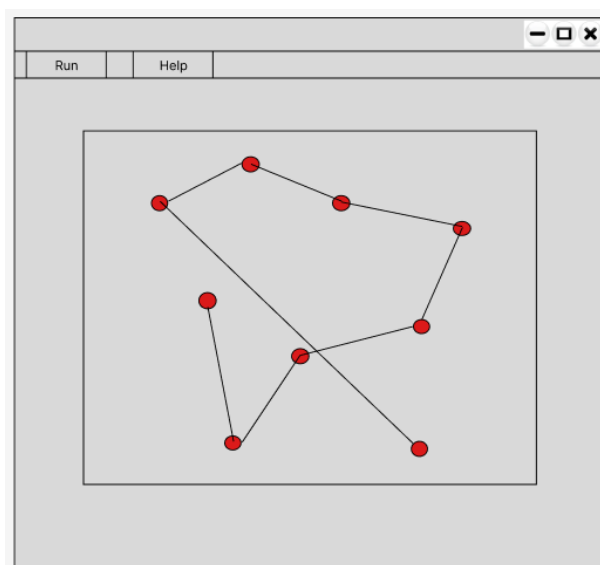
For the first implementation of the GA for constraint satisfaction, I decided to work on the travelling salesman problem. The concept of the overall project is to build a general program for constraint optimisation but for the first term I wanted to work on one constraint problem in this case TSP.

#### First Prototype



As you can see from my first proof of concept, I was slightly overambitious and underestimated the amount of time to become accustomed to the technologies and tools used. I decided to integrate the graphical interface with the actual program. Here you can see that the TSP visual appears on the left side of the screen and the distances plotted appears on the right side. There is a section on the menu called files which is where you would select the routes or paths.

#### Next prototype



For the next prototype I decided I wanted to go a simpler route and just display the TSP on the GUI. As you can see it is still quite interactive it has buttons how help navigate. It also shows visualisation in real time allowing the user to be able to understand and truly examine how the ga works with the travelling salesman problem. Here we have removed the distances and simplified the layout of the project.

As time progressed, I ended up separating the GUI and the visualisation as I found that pygame was much better for the visualisation. But I found that integrating pygame in to a Tkinter window is quite difficult and would be time consuming the only solution I could find was possibly using a thread.

Before creating my project, I started by experimenting with genetic algorithms firstly creating a simple genetic algorithm to solve an expression problem. Following this I started working on a genetic algorithm for the travelling salesman problem in python and doing research such as plotting graphs to gain a much deeper understanding of how it works.

### Genetic Algorithm For TSP

```

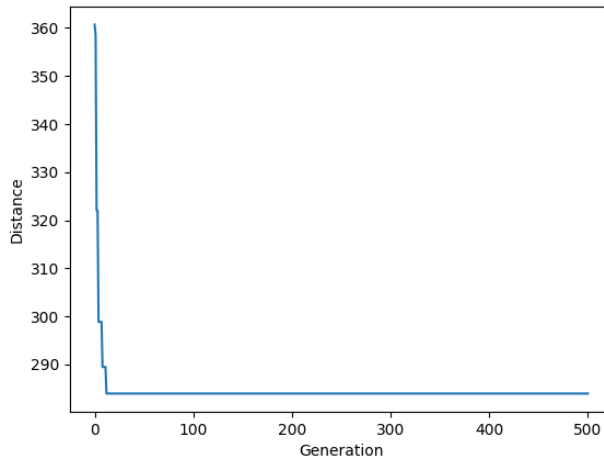
Gen = 496 curr distance = 314.9132071756672
Gen = 497 curr distance = 314.9132071756672
Gen = 498 curr distance = 314.9132071756672
Gen = 499 curr distance = 314.9132071756672
Final distance: 314.9132071756672
[Finished in 52.7s]

```

shows current distances and generations

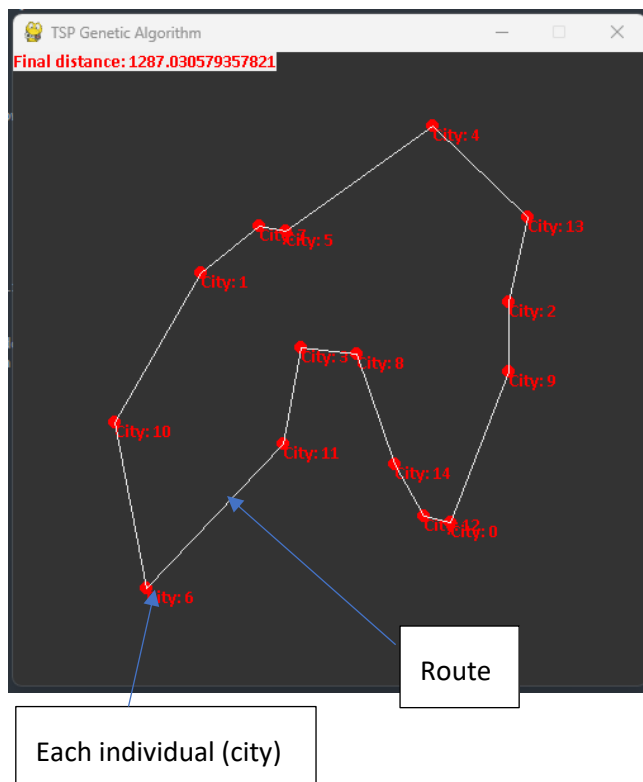
Final distance

The Genetic algorithm worked by using the processes of evolution to try to optimise the TSP.



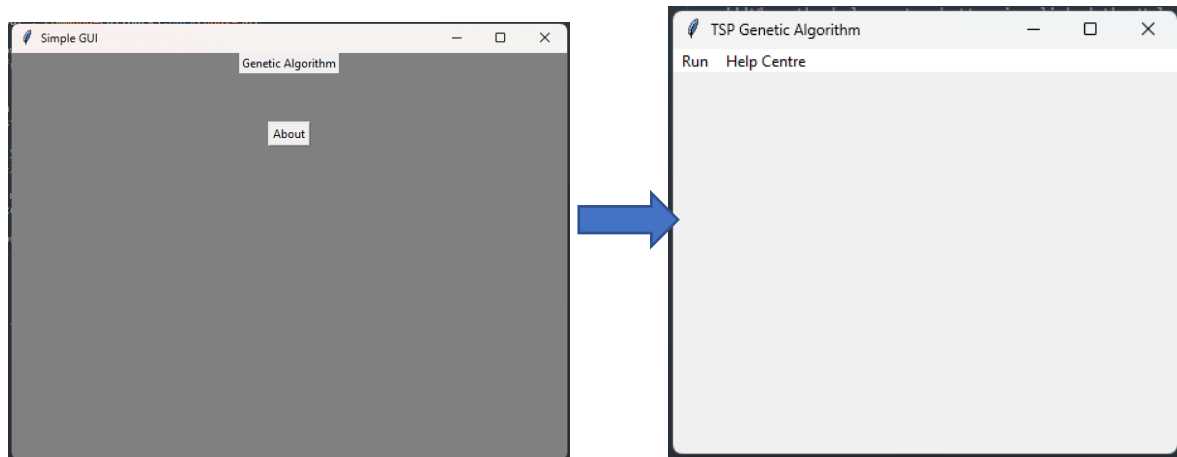
The conclusions I came to whilst doing this were that the higher the generation size the higher the time need to calculate distance and the higher the final distance is. The mutation rate also affects the distance as the higher the mutation rates the greater the final distance is. Whilst doing this I found that using biological notation helps further understanding.

### Pygame



For the visualisation of the project, I decided to use the python module known as pygame, this is often used to make games and visualisations. As you can see the visualisation makes use of shapes such as circles and lines to plot the travelling salesman problem. I also displayed the current destinations going to the final destination in the top right of the screen. The calculations and the process of finding the shortest distance used the basic concepts of genetic algorithms such as the fitness function and crossover.

### GUI Prototype



My GUI has changed quite a bit from my initial prototype, this is due to experimentation and actually working through designs using agile methodology. My first GUI prototype had both the visualisation and the GUI; it was more interactive. But as time progressed, I realised that creating the travelling salesman problem and visualising it in Tkinter was a much harder task then I realised and would take a lot more time. For the interim report I wanted to have something tangible to present as a result I opted to instead use pygame for the visualisation as it is much easier to plot objects on the screen and move them then it is in Tkinter. Following this I decided it would be best to have a simple GUI in Tkinter where you can run the program from and look for help. This resulted in me scaling back my prototype making it much better to work with the different libraries. Also, whilst research I found that the integration between Tkinter and pygame would also not be contributing to why I chose the scale back from my prototype. Further evidence of software engineering such as TDD and agile is proved with the progression of my prototypes.

## Chapter 4: Experimentation and Results

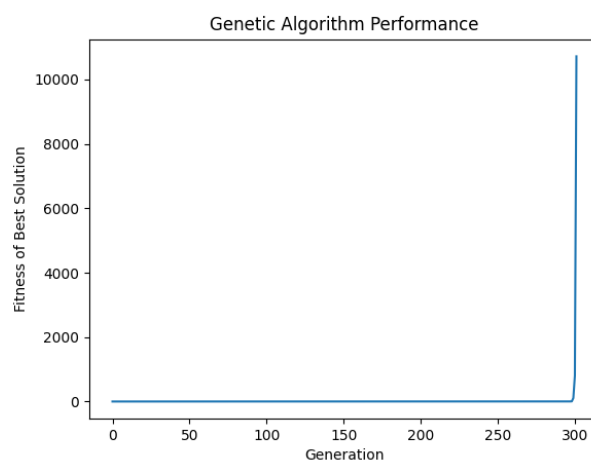
### Test different optimization problems:

#### Simple GA

To begin the project, I decided to first take a look at a simple genetic algorithm to find a solution that satisfies the equation " $6x^3 + 9y^2 + 90z - 25 = 0$ " in the function foo below.

```
# The function to be optimized
def foo(x,y,z):
    return 6*x**3 + 9*y**2 + 90*z - 25
```

It works by generating an initial population of 1000 solutions and then evaluate the fitness of each solution in the population by calculating the absolute value of the inverse of the solution's distance from the target value. If a solution satisfies the target value, a high fitness value of 99999 is returned. The solutions in the population are then ranked based on their fitness, with the best solutions at the top of the list. Later a new generation of solutions is created by mutating the elements of the best solutions. This is done by selecting one of the x, y, or z values from the extracted elements of the best solutions and multiplying it by a random value between 0.99 and 1.01. Lastly, we set the current population to the new generation, and repeat the process until a solution satisfying the target value is found, or a maximum number of generations (10000 in this case) is reached.



The plot shows that the algorithm's performance remains relatively constant for the first 300 generations, with small fluctuations in the fitness of the best solution. However around the point 300 generations the fitness of the best solution suddenly improves significantly. This indicates that the algorithm has found a solution. From this point, the fitness of the best solution continues to improve until it reaches the target value, at which point the algorithm then terminates.

Figure 13: GA simple program plot

We can surmise that the sudden improvement in the fitness of the best solution may be due to a lucky mutation which resulted in a solution that is closer to the target value. We know that genetic algorithms are stochastic optimisation algorithms which means that the performance can be somewhat dependent on the initial population and the random mutations that occur during the programs execution.

## Genetic Algorithm For TSP

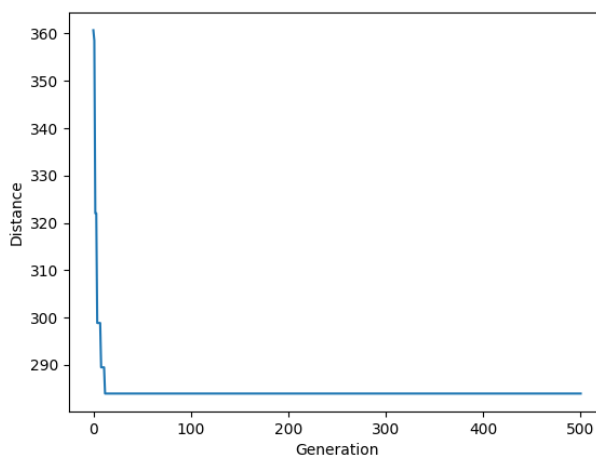
Following this I started working on a genetic algorithm for the travelling salesman problem in python and doing research such as plotting graphs to gain a much deeper understanding of how it works.

This program works by having a city with coordinates, a fitness function is applied which calculates the fitness of the routes (i.e., how good the route is) based on the total distance travelled and other methods to generate an initial population, rank the routes based on their fitness, select parents, and create a new generation of offspring. The ga repeatedly applies these to the population over generations to try to improve the fitness. To test the genetic algorithm, the code generates a random list of 10 cities and then plots the improvement in the fitness of the routes over 500 generations with a population size of 100, the best size of 20 (the number of routes to preserve as the best from each generation), and a mutation rate of 0.01.

```
Gen = 496 curr distance = 314.9132071756672
Gen = 497 curr distance = 314.9132071756672
Gen = 498 curr distance = 314.9132071756672
Gen = 499 curr distance = 314.9132071756672
Final distance: 314.9132071756672
[Finished in 52.7s]
```

shows current distances and generations.

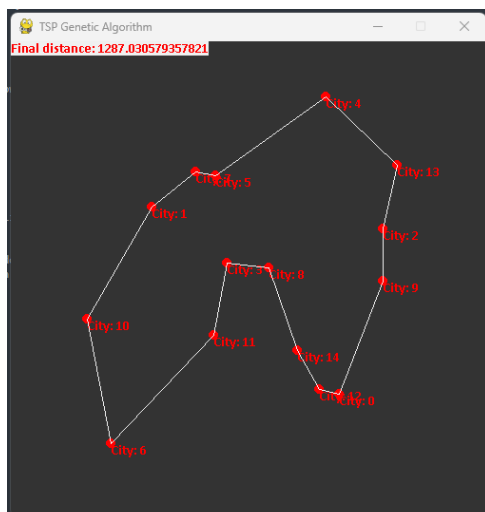
Final distance



The conclusions I came to whilst doing this were that the higher the generation size the higher the time need to calculate distance and the higher the final distance is. The mutation rate also affects the distance as the higher the mutation rates the greater the final distance is. Whilst doing this I found that using biological notation helps further understanding.

Figure 14: GA TSP program plot

## TSP Visual



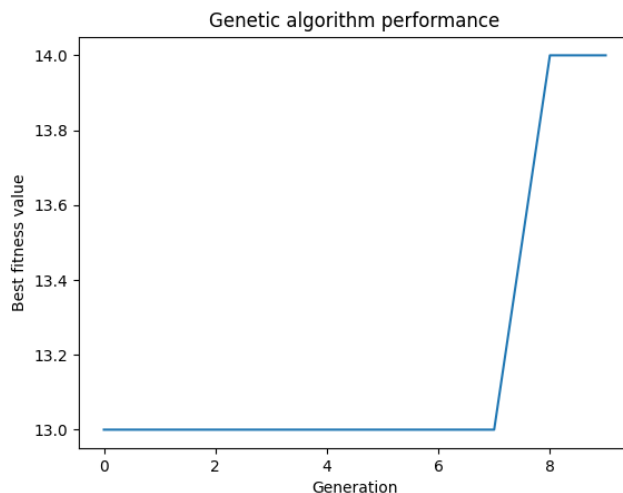
For the visualisation of the project, I decided to use the python module known as pygame, this is often used to make games and visualisations. As you can see the visualisation makes use of shapes such as circles and lines to plot the travelling salesman problem. I also displayed the current destinations going to the final destination in the top right of the screen. The calculations and the process of finding the shortest distance used the basic concepts of genetic algorithms such as the fitness function and crossover. For this program as the number of generations increased so did the amount of time needed to solve the problem thus showing that time constraints do really affect Ga's, but it did solve the problem without to many prompts.



## GA for Knapsack Problem

The next CSP I decided to try and solve with a genetic algorithm was the 0-1 knapsack problem it is a problem that I have come across while learning algorithms and complexity.

The genetic algorithm works by initialising a population of potential solutions in this case chromosomes with random values (0 or 1) . Iteratively evolving the population by selecting parents, performing crossover and mutation . Then evaluating the fitness until a stopping condition (in this case, a fixed number of generations) is met.

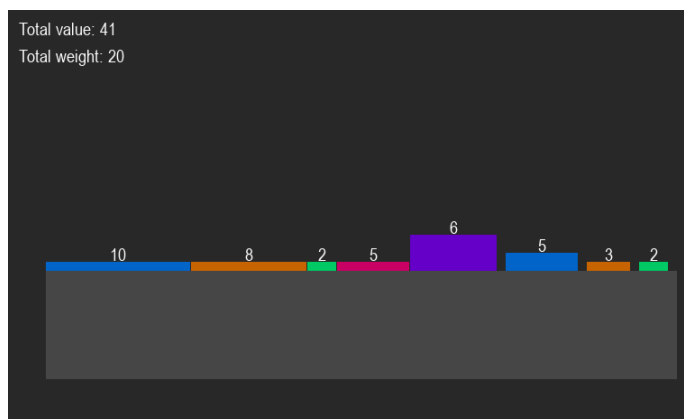


The plot shows the fitness score of the best solution found by the ga for each generation. The x-axis represents the generation, and the y-axis represents the fitness score. The fitness score is initially stagnant, but starts to increase after the 7<sup>th</sup> generation and eventually levels off. This suggests that the ga is able to find the better solution over time and converges to a good solution.

Figure 15: GA Knapsack plot

I just wanted to note that while going through these tests I am finding that lucky mutation and other random events are occurring which means that the results can be different or often random at times.

## Knapsack Visualisation



Solving knapsack problem with pygame visualisation. An aspect of the project is that I needed to create a simulation and visualisation of different problems. From experimenting with this program I have found that although genetic algorithms can be used to solve CSPs they may not be efficient in solving the problem. If given a time constraint it may not be possible to solve the problem in a required time this was

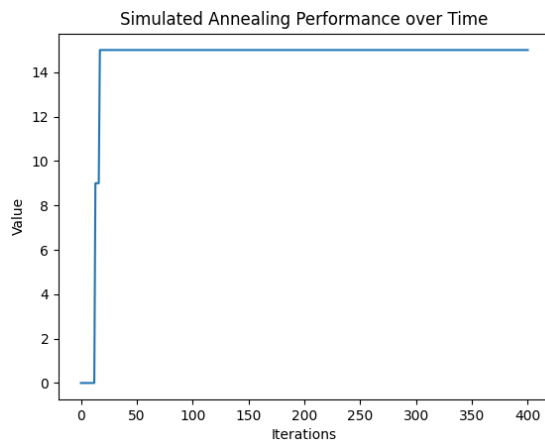
often the case, I found when trying to visualise the problem I found that some programs just kept running and would not end. Although Gas can be used to solve problems, they may not be the best or most optimal solution.

### **Compare with other Optimization Methods**

After this I decided to take a CSP problem and compare the performance of GA with other optimization techniques, such as Simulated Annealing. This will help to understand the strengths and weaknesses of GA in relation to other methods.

### **Simulated annealing Knapsack Problem**

This simple implementation uses the simulated annealing algorithm to generate random solutions to the 0-1 knapsack problem and iteratively improve them by flipping random bits. Then accepting the new solution based on an acceptance probability that depends on the temperature of the system.



The plot shows that the simulated annealing algorithm was able to improve the solution value over time, with the value gradually increasing and reaching a peak around iteration 40, before levelling off and remaining constant for the remaining iterations. The final best solution value obtained was 15.

*Figure 16: Simulated annealing Knapsack plot.*

From looking at both plots it can be seen that the convergence speed for the simulated annealing problem is actually faster than the genetic algorithm. This could be because it focuses on refining the search space around the current solution. In contrast ga's require a larger number of iterations to explore a wider search space to find the optimal solution.

## Function optimisation

To further extend my project and make it more general I decided to experiment and apply a genetic algorithm to an optimisation problem, in this case trying to optimise the Rastrigin function.

### Rastrigin function

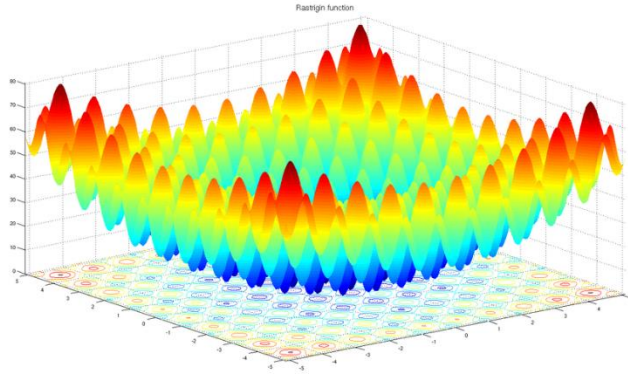


Figure 17: Rastrigin Function

The Rastrigin function [16] is known as a non-convex, multivariate function used as a performance test problem for optimisation algorithms. The formula for the Rastrigin function is as follows:

$$f(\mathbf{x}) = 10d + \sum_{i=1}^d [x_i^2 - 10 \cos(2\pi x_i)]$$

Where  $d$  is the number of dimensions,  $x_i$  is the value of the  $i$ -th dimension of the input vector  $\mathbf{x}$ , and 10 in this case is a constant parameter that controls the “roughness” of the function. The Rastrigin function is well-known for having many local minima, making it a challenging problem for optimization algorithms. Its global minimum is at  $f(\mathbf{x}) = 0$ , which occurs at  $\mathbf{x} = (0, 0, \dots, 0)$ . The Rastrigin function is often used as a benchmark function for comparison of performance for different optimization algorithms. The function is used to test the ability of optimization algorithms to converge to the global minimum in high-dimensional spaces.

On an  $n$ -dimensional domain it is defined by:

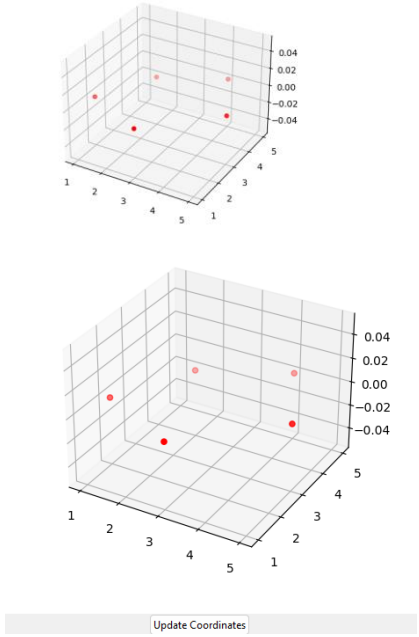
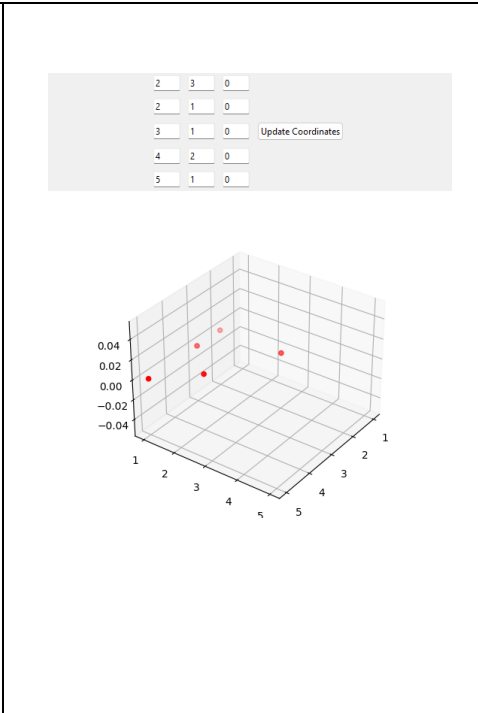
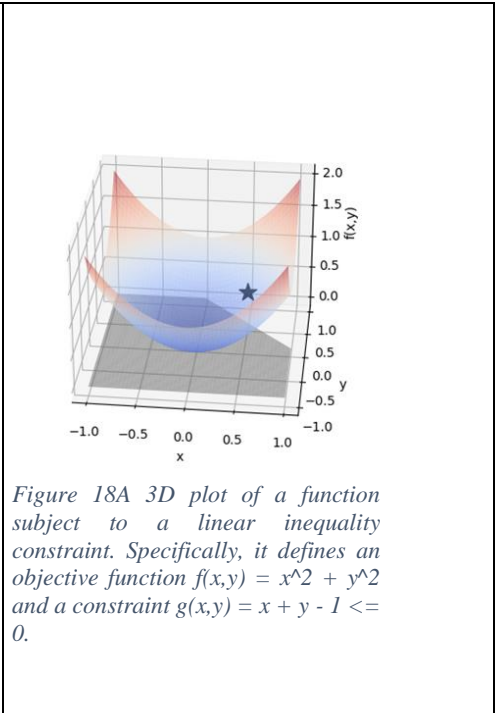
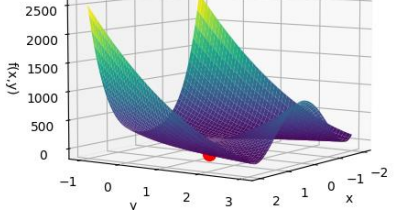
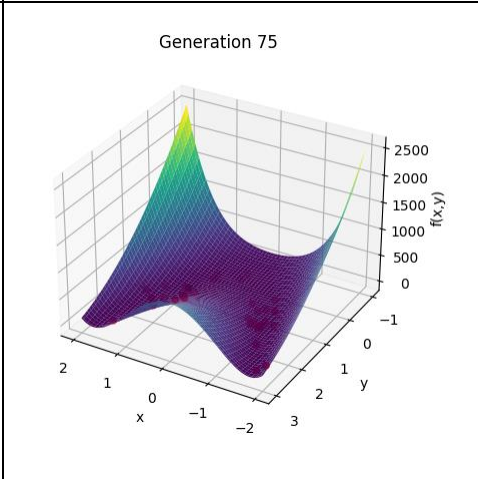
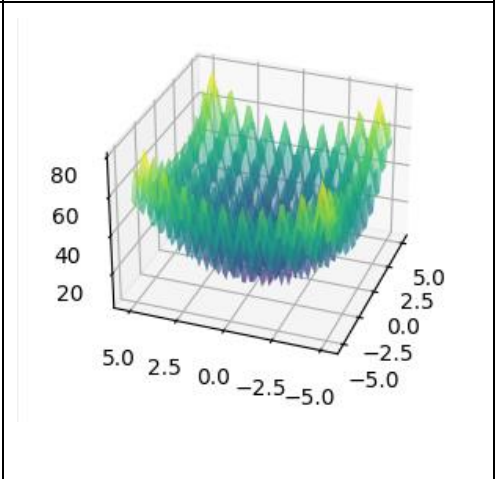
$$f(\mathbf{x}) = An + \sum_{i=1}^n [x_i^2 - A \cos(2\pi x_i)]$$

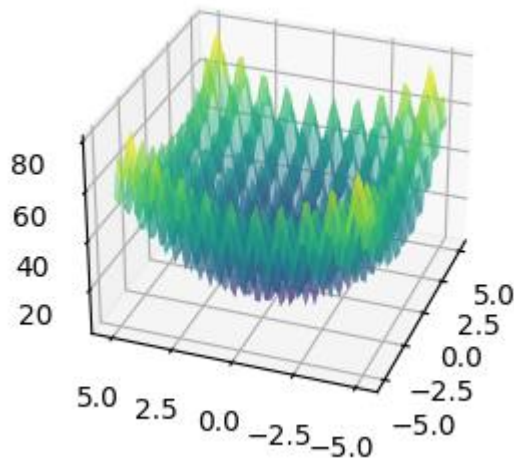
where  $A = 10$  and  $x_i \in [-5.12, 5.12]$ . There are many extrema:

- The global minimum is at  $\mathbf{x} = \mathbf{0}$  where  $f(\mathbf{x}) = 0$ .
- The maximum function value for  $x_i \in [-5.12, 5.12]$  is located around  $x_i \in [\pm 4.52299366\dots, \dots, \pm 4.52299366\dots]$ :

For this test I decided to use a 3D model to find the global minimum of given function in this case the Rastrigin function. I decided to make it in a way where I could experiment with various GA parameters such as population size, crossover rate, mutation rate and the type of selection method (between tournament and roulette) to see how they affect the performance of the GA.

The design process for this program was quite strenuous.

|  |  |  |
|--|--|--|
|    |    |  <p>Figure 18A 3D plot of a function subject to a linear inequality constraint. Specifically, it defines an objective function <math>f(x,y) = x^2 + y^2</math> and a constraint <math>g(x,y) = x + y - 1 \leq 0</math>.</p> |
| <p>I learned to create a 3D model in Tkinter using Matplotlib and updated the coordinates with a button to animate the points.</p> | <p>Then learned how to update the plot with input coordinates.</p>   | <p>I then proceeded to learn how to plot function on a 3d plot using matplotlib. This is a visual of an of optimization problem.</p>   |
|    | <p>Generation 75</p>   |   |
| <p>Following this I then went on to learn to apply a genetic algorithm to optimise the function.</p>                               | <p>Next, I applied a genetic algorithm to optimize the function and create a dynamic plot. Although this version didn't work perfectly, it taught me how to update the plot and utilize a genetic algorithm for optimization. My goal is to make the optimization interactive by allowing users to input values for the genetic algorithm.</p> | <p>Then finally using the logic I learned applied it to the Rastrigin function using some ideas from <a href="#">MATLAB</a>. Here is the function: "20 + x1 * x1 - 10*np.cos(2 * np.pi * x1) + x2 * x2 - 10 * np.cos(2 * np.pi * x2)", -5.12, 5.12</p>   |

**Experiment with param →**

What I learned from this is that the optimization process with genetic algorithms is highly dependent on the input parameters. Even small changes in these parameters can have a significant impact on the optimization results. It's important to carefully consider the selection of these parameters to achieve the desired results. Through this process, I gained a better understanding of the importance of parameter tuning in optimizing algorithms, as well as the challenges involved in finding the optimal combination of parameter values.

Disclaimer- I was going to add a section detailing the different types of selection from roulette to tournament however, due to complexity and time constraint this was not possible.

## Chapter 5: Final Deliverables and Conclusion

### Travelling salesman Problem

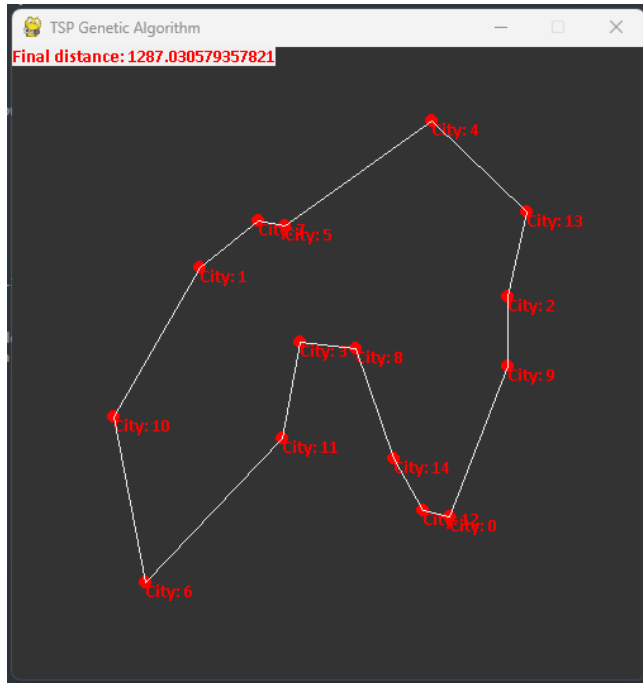


Figure 19: Visual for TSP GA

#### Explanation:

The program above is an implementation of a genetic algorithm to solve the travelling salesman problem a classic problem in computer science and optimization. As stated, earlier the TSP is described as the task of determining the shortest path that passes through a specified group of cities, returns to the starting city, and visits each city precisely once.

The program starts by defining two classes: City and Fitness Function. City represents a city in the TSP with its x and y coordinates and a label to identify them. The Fitness Function class represents a possible solution to the TSP, which is a sequence of cities. It includes a method to calculate the distance travelled in the route.

The main function of the program carries out the genetic algorithm to solve the TSP. It first initializes a population of random solutions (Fitness Function instances). Then, it enters a loop where it ranks the population by fitness (by the distance of the routes), selects the best individuals to create a new generation by crossover, and repeats until the desired number of iterations is reached. Whilst also displaying the problem being solved the final distance is also being calculated and displayed at the top. The code is also divided into three parts: Model, View, and Controller using the MVC architecture.

## Knapsack problem visualisation

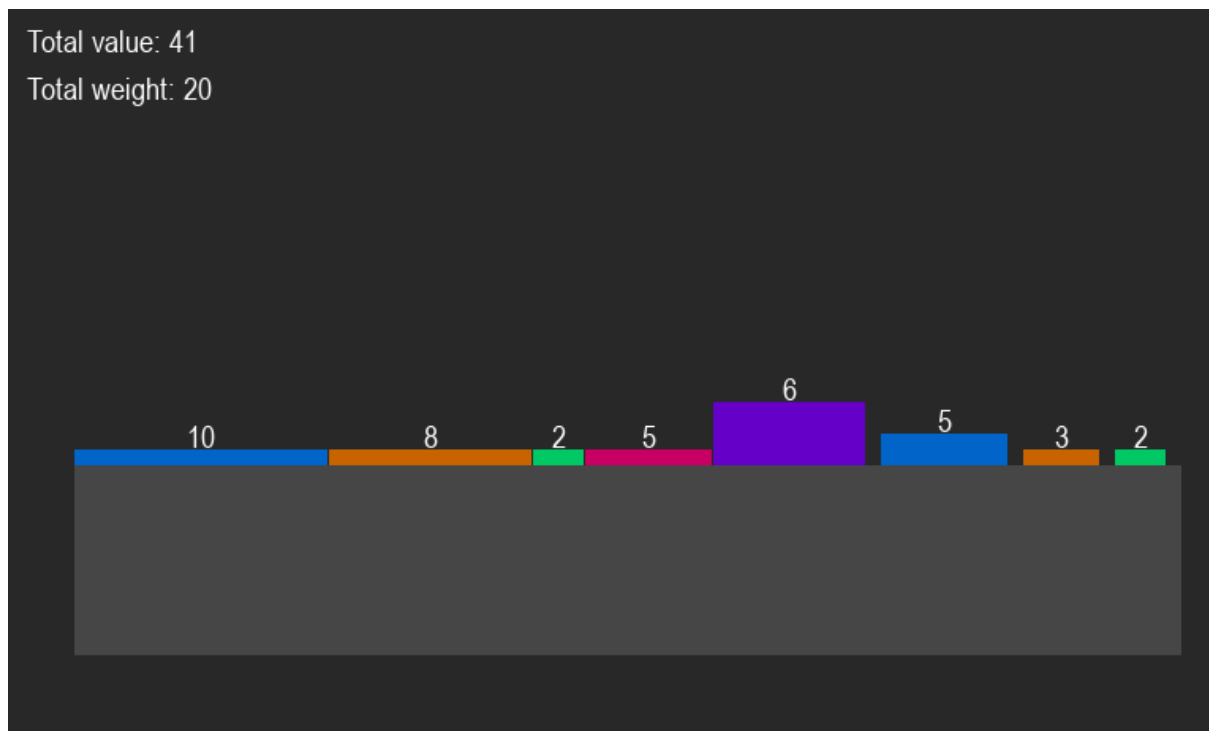


Figure 20: Visual for Knapsack GA

#### Explanation:

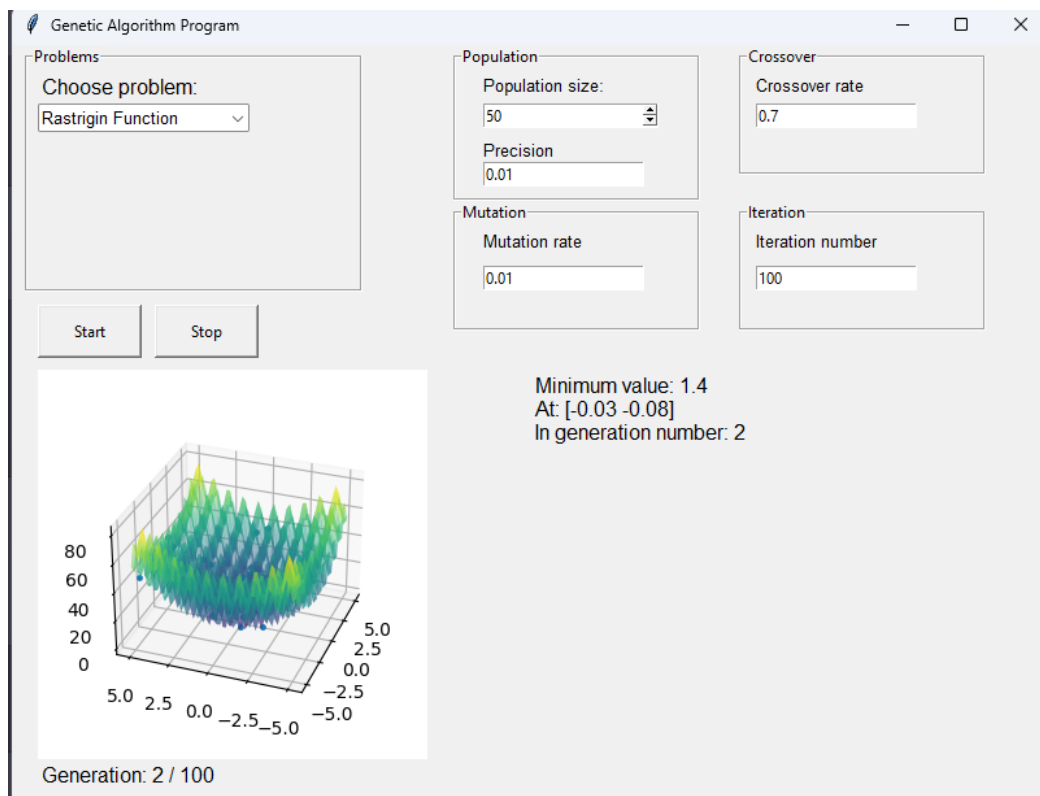
This program displayed above is an implementation of a genetic algorithm to solve the Knapsack problem which is a combinatorial optimisation problem. As stated earlier the knapsack problem is described as a problem where we are given a set of items each with a weight and a value and a knapsack (bag) with a maximum capacity. The goal of this is to find the subset of items that maximises the value of the items in the knapsack, without exceeding the maximum capacity.

Here in this the program, it initialises with a set of items each with their respective values and weights and a capacity for the knapsack. Following this it then generates a population of solutions, where each solution represents a subset of the items. The genetic algorithm then evolves the population by selecting parents based on their fitness (the total value of the items in the subset) and applying crossover and mutation operators to generate offspring.

The fitness function calculates the total value of the items in a subset and returns 0 if the total weight exceeds the capacity. The program is using a binary encoding scheme to represent the solutions to the knapsack problem. Each item is either included in the knapsack therefore represented by 1 or conversely excluded so represented with a 0 in the solution. The binary encoding of the solution is represented by a list of 0's and 1's, where the  $i$ -th element of the list represents whether the  $i$ -th element of the list is either included (represented by 1) or excluded (represented by 0).

The selection function implements the roulette wheel selection (the different types of selections are discussed [GA background]) to select the parents based on their fitness. The crossover function uses a random single point crossover to then generate two offspring from the two parents. The mutation function applies a mutation operator to each gene so in this case each item in the solution with a particular probability. The program following this draws the best solution found so far as well as the total value and weights. Each item is represented by rectangle with the larger rectangle on the bottom representing the sack/bag.

## Interactive Genetic Algorithm Problem



## Explanation:

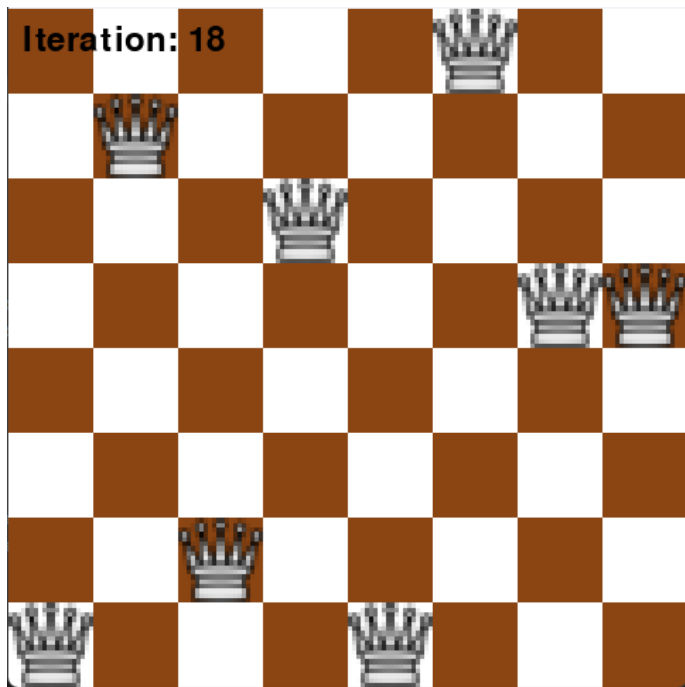
This program is a GUI for running a genetic algorithm (GA) to optimize the Rastrigin function. The program is built using the tkinter library for creating the GUI and matplotlib for plotting. The GUI for the interactive based simulation aspect of the project, that enables the user to interactively view and control GA optimization for function optimisation. There are different input boxes which also the users to input and test different conditions for the GA as well as the problem. There are also buttons which allow the program to be initiated and to be terminated. I chose to create a simple readable GUI to allow users to instantly be able to use and understand it. Below you can see the plot for the function we want to optimise and to the right the number of generations as well as the current minimum value calculated

**Overview of how the program works**

This program is a graphical user interface (GUI) for running a genetic algorithm to solve optimization problems. It uses the Tkinter library for the GUI layout and widgets, and the matplotlib library for displaying surface plots. When the user starts the algorithm, it iterates through selection, crossover, and mutation operations, updating the best solution and generation number along the way. The algorithm continues until the maximum number of iterations is reached or the user stops it. Users can interact with the GUI by selecting different problems, changing parameters, and observing the surface plot and results displayed in the interface.



## N-queen Problem

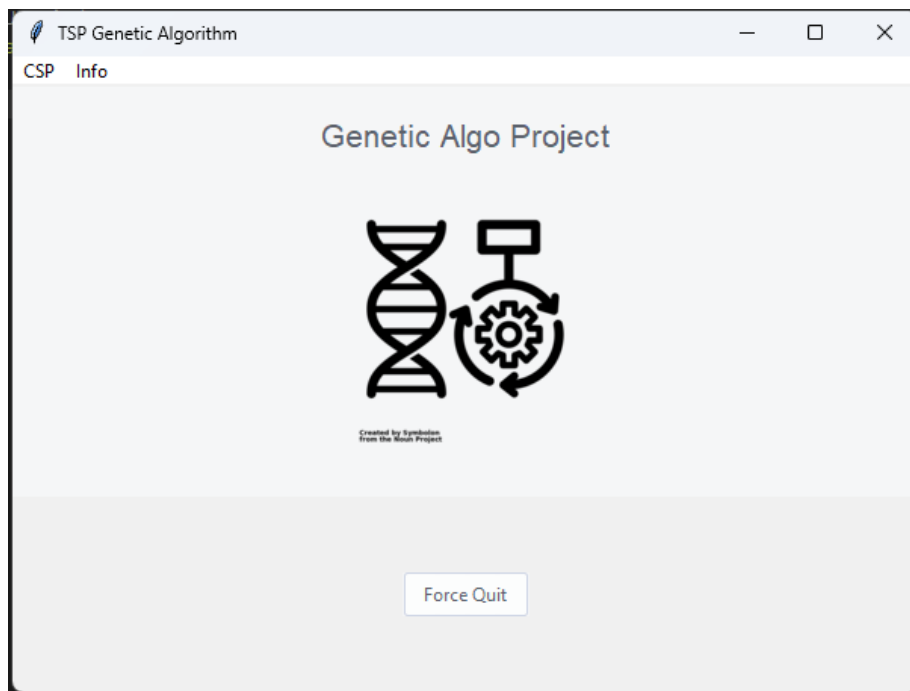


This program solves the N-Queens problem using a genetic algorithm in Python and visualizes the solutions with the Pygame library. The algorithm iterates through a process of selection, crossover, and mutation to find a solution where no queens threaten each other on an  $N \times N$  chessboard. Elitism is used to preserve the best solutions found so far. The performance and efficiency of the program may be affected by various factors such as population size, mutation rate, and the number of iterations.

During the program execution, the Pygame library is used to visualize the progress of the genetic algorithm in real-time. The chessboard and the current best solution are drawn on the screen, with queen icons placed at the positions determined by the algorithm. The fitness function measures the quality of each solution, and the selection function chooses parents based on their fitness for crossover. The crossover and mutation functions generate new offspring, which replace the least fit individuals in the population.

The program incorporates elitism to preserve a specified number of the best solutions, ensuring they are not lost due to selection, crossover, or mutation. This strategy helps maintain high-quality solutions in the population and promotes convergence towards a solution to the N-Queens problem. The main loop of the program continues until the maximum number of iterations is reached or a satisfactory solution is found. Throughout the runtime, the user can observe the algorithm's progress as the Pygame window displays the chessboard, queens' positions, and iteration count. It is essential to note that the program's performance and efficiency may vary depending on factors such as population size, mutation rate, and the number of iterations. The genetic algorithm may not always find an optimal solution within a limited number of iterations, but adjusting these parameters can potentially improve the algorithm's performance.

## GUI

*Figure 21: GUI*

In order to use this image, I needed to give credit to the creator which is "[genetic algorithm](#)" icon by Symbolon from Noun Project CC BY 3.0.

GUI modifications: For the GUI I started to focus more on usability and ease of access. The GUI features buttons and as well as a menu bar to allow the user to able to navigate the program and access different parts. The buttons are labelled clearly using concise descriptive text to inform the user of their purpose. The menu bar is organised logically, and related options are grouped together. A sub menu is also used to further organise options and prevent the menu bar from becoming too clustered. Plain contrasting colours to make the text easier to read and larger text sizes or bold text are used to further enhance the readability of the information. Furthermore, on the info section contrasting colours are used for visually impaired users. The info section also discusses how to use the program the instructions are clear and concise whilst using simple language and avoiding technical jargon.

## 5.1 Reflection of first term

### What I did?

For the first term I decided to work on and apply a genetic algorithm to one constraint optimisation problem. The problem I decided to explore was the travelling salesman algorithm by mathematician William Rowan Hamilton. I started by conducting in-depth research on genetic algorithms and learning how they operate, then I researched constraint optimisation. Following this I proceeded to make a simple genetic algorithm to find specific numbers to complete an expression. With more knowledge I then decided to try and create a genetic algorithm for the travelling salesman problem and explore the how distance improved overtime along with each generation. Following this I planned to create a visual based genetic algorithm in python using similar principles explored in the aforementioned genetic algorithm. This however did not fully go to plan as I underestimated the amount of time it would take as well as the workload not only for the project but other modules that were rigorous.

### What didn't I do?

I did not manage to follow the intended project plan closely. At times I strayed from the project plan and decided to focus more time into certain aspects of the project. For example, I didn't account for how hard it would actually be to create a visualisation for the algorithm and how some tools may have limitations when creating the visualisation. Resulting in not the best version of work possible. Furthermore, I admit I did not adequately follow the intended method of uploading working as some parts of the project were just genuinely hard to complete and I did not want to commit random components that didn't make sense.

### How is the current project different from proposed project?

The current project slightly differs from the intended project as with my sketches I intended for the latter part of the project to be fully interactive allowing the user to be able to select a generation size with a graphical interface and see how the different traversals. This is due to how hard it was trying to use the already created genetic algorithm for a reference point for the visualisation as certain functions were a bit too complex to be able to carry out in the program with limited time doing this would not be feasible. Which is why the code for the visualisation will look quite different reflecting time constraints. All in all, though I am quite proud of where I am currently as I have a greater understanding of genetic algorithms and their use in constraint optimisation. Furthermore, this will serve as a reference point for the future as I will be able to gauge how much time it will take to carry out certain parts of the project.

## 5.2 Second term aims

### Final Deliverables

- Research constraint optimisation in a broad sense and how it can be further extended
- Write a survey of optimisation methods (carryout survey)
- Investigate other problems for optimisation problem
- Compare different optimisation methods applied to the same problems
- Implement the broader Constraint optimisation aspect of the project
- Look to add other optimisation methods
- Continue to integrate GUI with genetic algorithm

## 5.3 Reflection overall

In general, the project went quite well in some aspects the project has come with many highs and lows with periods of me having a good idea of how to do things to having no clue on how certain topics work. As such I have had to learn in this project and to improve on. The parts I did well on were those that involved some sections of biology but also sections that involved optimisation. This is because although not to this degree in some module I did go over some topics that were quite useful in the project. Trying to find the intersection of biology and computing and applying it was quite fun and showed me the different aspects technology can be used in real life. I think I more so enjoyed this as my course is computer science and so I hadn't had much experience and prior knowledge for the biology so researching this and learning was incredible useful and interesting.

Other aspects of the project that went well was the structure I put in place with my plan making more loose but still focused. In this aspect I made sure to follow the project plan and expand on particular section where necessary for example investigating other optimisation problems but doing this in more detail. Although others may not have found it useful to be it proved to be very helpful as I meant each week I had an understanding of what I needed to be completed and the time constraint I had. The scope and loose layout of the plan proved to be helpful during the last stretch of my project as the programs I was working on were quite difficult due to the complexity of the project and the features I wanted to implement to make sure the project met the criteria. I knew when making the plan that the project could be subject to changes as I progress and learn and thus making a more loose but ridged plan served to be useful.

However, I must be honest and admit that unfortunately some parts of the project did not go as expected. One of the problems I faced was time constraint and time management. Despite the project plan helping I did not well anticipate the different external factors that may affect the project as well as the sheer size and complexity of the project. An example of this was not factoring in properly my other modules and not anticipating deadlines and assignments that may interrupt my plans which resulted in me having less time to work on my project at some times. This happened quite a lot as I am doing masters and the modules like data analysis were quite rigorous proving to be very stressful at times resulting in me feeling quite panicked and drained. In retrospect I would have create a more detailed project plan for the first time in particular to account for other factors that may affect the project like getting sick or having to spend extra time studying a particular concept like kernels. This would make sure that that I am more on track and can allocate my time more accurately across all classes and external events like sports (part of a sport team).

One other aspect of my project that didn't go quite well was the maintenance of my project diary. In the initial phase, I was able to keep the diary up to date by regularly adding new information and making improvements. However, as the project progressed into the second term, it became increasingly challenging to update the diary regularly. The workload became so daunting, and I often forgot to update the diary due to time constraints. This issue further highlights the challenge I faced with time management during the project, as I struggled to prioritize between various tasks and responsibilities. Overall, the maintenance of my project diary was impacted by the workload and time management challenges that I encountered during the project.

What I learned about doing a project is that it takes quite a bit of persistence and foresight. If I had been more proactive in anticipating the external factors that could affect my project, I would have been better equipped to handle the challenges that arose. Upon reflection I now understand that project planning should involve an assessment of potential risks and uncertainties, and a possible contingency plan should be developed for any unforeseen circumstances. Overall, I learned that a successful project requires careful planning, effective communication, and a flexible approach to adapt to changes in circumstances. By being persistent and proactive in addressing potential challenges, I could have had less sleepless nights and had more time to work.

## Chapter 6: Professional Issues

### What is the meaning of professional issues?

To begin we know that professional issues refer to the ethical, legal, social, and professional considerations that may occur when computing technologies and practices interact with society. These issues can take place in many different areas of computing such as software development, data management, network security and user interface design. So, examples of professional issues in computing that relate to my project include the concerns about the responsible use of artificial intelligence and machine learning, the privacy and protection of personal data, appropriate use of copyrighted and open-source software and the ethical issues surrounding the insurgence of new technologies like virtual reality and augmented reality. Professional issues are important as because they can have a significant and lasting impact on companies and society as a whole. Companies and corporations must be able to set a list of moral and ethical codes that employees must follow, often some professional values and ethics dictate the standard of profession. Addressing these issues takes a deep understanding of the complex relationship between technology and society more so the commitment to responsibility and ethical practices. It also requires the development of policies, guidelines, and standards that can help to ensure that technologies are developed, deployed, and used in ways that benefit everyone.

### Usability

An aspect of my project that was particularly concerning was the usability particularly the ease of use and colour contrast chosen initially. To be honest the usability whilst considered in the proof of concept was not as well as I thought it should be. I did not really consider the use of colour contrast and large bolded words, so I did not really consider the accessibility of users for example those that might be visually impaired (which is quite weird as I am somewhat) but rather wholly focused on the implementation. The usability is important as the ability to navigate and understand the program is important. In order to rectify this, I included a menu bar to ease the complexity and allow the user to be able to easily navigate the program. Furthermore, I include a theme to the GUI, an image denoting the program and made the name of the program bold to allow user to be able to recognise it. Thus, making more easy for the user to be able to recognise the program and also understand different aspects of it. Following this there is also a section on the menu bar called “info” when clicked it gives instructions on how to run the program. Ways that could have improved the usability of within the proof of concepts would be to more cautious of the intended use and audience for my design and be conscious of the user that may or may not have knowledge of the program or impairments that may affect their usability.

### licensing

Licensing refers to the legal framework that governs the modification, use and distribution of forms of intellectual property. It is a serious issue in the computing industry as we know software is often protected by copyright law and different licensing agreements which are used to define the terms and conditions under which software can be used or distributed. Understanding software licensing is important for developers and organizations because it can impact how software is used, shared, and distributed. It is important to carefully read and understand license agreements before using or distributing software to ensure that you are in compliance with the terms and conditions of the license. A particular example of where I wanted to use an image for my GUI from Noun Project. The image I chose was listed as free but only under the condition that I give credit to the creator using a particular format including this “license type (CCBY3.0)” and a hyper link when referencing. These were the term and conditions set in place in order for me to use the image. This affected my project

as it made more aware of licensing and the necessity of meeting terms and conditions. It was of ethical and practical importance as if I did not comply, I would not be given the authorisation to use the image and therefore may be liable to some consequences. For some aspects of the project with this in mind I got around the issue by creating my own images and design.

### **Plagiarism**

A professional issue that I had to consider during the project is plagiarism. Plagiarism could arise in multiple different ways for example if there were some sections of the report that I had to research and required referencing from websites and books and I did not end up referencing them in the report this would be deemed a form of plagiarism as it would ethically be deemed as a form of theft. By stealing the ideas and texts of others and pretending they are yours, you are stealing someone else's intellectual property. To avoid this, I decided to address the issue by using citation in my work and code to state that I have used some ideas as well as a bibliography to store different references. Another example of this so when I first learned about the TSP, I was not very familiar so I used ideas from code by Eric Stolz but made sure to credit him where used, had I not done this it would have been in breach and therefore considered plagiarism.

### **An example from the public domain of what can happen when professional issues are not properly addressed.**

An example from the public domain of what can happen when professional issues are not properly addressed is the case of the Therac-25 medical radiation machine. The Therac-25 was a computer-controlled radiation therapy machine that was developed in the 1980s and was used to treat cancer patients. In the 1980s, several patients who were treated with the Therac-25 were given massive overdoses of radiation, causing severe injuries and even death in some cases. Later it was discovered that the overdoses were caused by a software bug in the machine's control software that caused it to deliver high doses of radiation without proper safety checks.

The Therac-25 case highlights the importance of addressing professional issues in the development and use of technologies, particularly those used in critical applications such as medical treatment. The incident was the result of a failure to properly test and validate the software that controlled the machine, as well as a lack of safety checks and controls for prevention of overdoses. This case led a significant change in the way that medical devices are regulated and developed, with increased emphasis on safety and risk management. Medical device manufacturers are now required to follow strict guidelines for the development and testing of their products, which includes rigorous software testing and validation processes. The Therac-25 case serves as a cautionary tale of what could happen if professional issues are not properly adhered to in computing and underscores the importance of responsible and ethical practices in the development and use of technology today. It gains relevance to the project as it is a real world problem involving technology to optimise a process similar to what I am trying to explore in the project.

# Glossary

**Selection:** chooses the parents, who will contribute to the population of the following

**Mutation:** subjects each parent to random modifications.

**Crossover:** combine two parents to create the next generation's offspring.

**Creational:** Used to create objects that can be separated from the systems that implement them.

**Structural:** Deal with coupling, used to connect numerous dissimilar objects into large object structures.

**Behaviour:** These patterns are concerned with algorithms and the assignment of responsibilities between objects.

**Architectural:** Describes a system's assemblies, layers, or environment

**MVC:** decoupled and they are responsible for the model, view, and controller

## Acronyms

CPS - Constraint Satisfaction Problem

GA - Genetic Algorithm

TSP – Travelling Salesman Problem

TDD - Test-driven Development

MVC – Model/View/Controller

OOP – Object orientated programming

# Bibliography

- [1] En.wikipedia.org. 2022. *Genetic algorithm - Wikipedia*. [online] Available at: <[https://en.wikipedia.org/wiki/Genetic\\_algorithm](https://en.wikipedia.org/wiki/Genetic_algorithm)> [Accessed 22 September 2022].
- [2] En.wikipedia.org. 2022. *Genetic algorithm - Wikipedia*. [online] Available at: <[https://en.wikipedia.org/wiki/Genetic\\_algorithm](https://en.wikipedia.org/wiki/Genetic_algorithm)> [Accessed 22 September 2022].
- [3] Cs.mcgill.ca. 2022. [online] Available at: <<https://www.cs.mcgill.ca/~dprecup/courses/AI/Lectures/ai-lecture05.pdf>> [Accessed 27 September 2022].
- [4] Holland, J.H. (1992) *Adaptation in natural and artificial systems: An introductory analysis with applications to biology, control, and Artificial Intelligence*. Cambridge: The MIT Press [Accessed 22 October 2022].
- [5] Rosenberg, N.A. and Nordborg, M. (2002) "Genealogical trees, coalescent theory and the analysis of genetic polymorphisms," *Nature Reviews Genetics*, 3(5), pp. 380–390. Available at: <https://doi.org/10.1038/nrg795>.
- [6] Stoltz, E. (2021) *Evolution of a salesman: A complete genetic algorithm tutorial for python, Medium*. Towards Data Science. Available at: <https://towardsdatascience.com/evolution-of-a-salesman-a-complete-genetic-algorithm-tutorial-for-python-6fe5d2b3ca35> [Accessed: December 4, 2022].
- [7] Buckley, G. (2021) Genetic drift - definition, examples and causes, Biology Dictionary. Available at: <<https://biologydictionary.net/genetic-drift/>> [Accessed: November 28, 2022].
- [8] Falkenauer, E. (1999) *Genetic algorithms and grouping problems*. Chichester: Wiley [Accessed 27 November 2022].
- [9] Goldberg, D.E. (2013) *Genetic algorithms in search, optimization, and machine learning*. New Delhi, India: Pearson.
- [10] Heest, M.van (no date) *Every shortcut for designers, centralized and searchable, Sublime Text shortcuts | All shortcuts for Sublime Text - Shortcuts.design*. Available at: <https://shortcuts.design/tools/toolspage-sublimetext/> [Accessed: December 6, 2022].
- [11] Ashlock, D. (2006) *Evolutionary computation for modeling and Optimization*. New York: Springer. [Accessed 27 November 2022]
- [12] Kryvokhyzha, D. (2018) *Genome evolution and adaptation of a successful allopolyploid, capsella Bursa-Pastoris*. dissertation. Acta Universitatis Upsaliensis [Accessed 21 November 2022].
- [13] van, L.P.J.M. and Aarts, E.H.L. (2010) *Simulated annealing: Theory and applications*. Estados Unidos: Reidel Publishing Company [Accessed 16 February 2023]
- [14] GERON, A. (2019) *Hands-on machine learning with scikit-learn, Keras, and tensorflow: Concepts, tools and techniques to build Intelligent Systems*. Beijing ; Boston ; Farnham etc.: O'Reilly. [Accessed 16 February 2023].
- [15] Nocedal, J. and Wright, S.J. (2006) *Numerical optimization*. New York, NY: Springer [Accessed 13 February 2023].



- [16] *Rastrigin function* (2023) *Wikipedia*. Wikimedia Foundation. Available at: [https://en.wikipedia.org/wiki/Rastrigin\\_function](https://en.wikipedia.org/wiki/Rastrigin_function) [Accessed: March 21, 2023].

# Appendix

## Timeline of the project plan

GENETIC ALGORITHM

4

### Timeline

My plan will be for the first term to look at constraint optimization and then apply it to one problem in this case being the TSP. Then to go with a wider approach to look at a ga's implementation on a wider scope.

#### Term 1

##### Week 2

- Create 2d prototype sketches for the proof of concept.
- Create layout of report
- Installation of necessary software and Ide to produce program, algorithm, and GUI. So far likely python and Tkinter.
- Experiment with creating GUI with sublime text

##### Week 3

- Begin research on Genetic algorithms and write the basis of the report.
- Draft template for GUI
- Begin implementing the GUI - a simple version

##### Week 4

- Research the implementation of Genetic algorithm
- Create a simple genetic algorithm

##### Week 5

- Work more on building genetic algorithm for TSP
- Write report on encoding various problems for GA's

##### Week 6

- Try to use Tkinter with genetic algorithm
- Begin research on Design patterns for write up e.g., MVC Design Pattern

##### Week 7/8

- Animating and simulations for genetic algorithms
- Write up on the theory of coalescence and genetic drift

##### Week 9

- Link GUI and Genetic algorithm together
- Write up remaining parts of the report to a good standard

##### Week 10/11

- Finalise TSP problem for project
- Finalise report

#### Term 2

I expect by term 2 that a lot of the project plan will change due to the size of the project, so I have purposely made this section more loose but still with detail.

##### Week 1/2

- Research constraint optimisation in a broad sense and how it can be further extended
- Start to describe the software engineering processes involved

##### Week 3

- Write a survey of optimisation methods (carryout survey)

##### Week 4/5

- Investigate other problems for optimisation problem
- Compare different optimisation methods applied to the same problems

##### Week 6

- Implement the broader Constraint optimisation aspect of the project
- Look to add other optimisation methods

##### Week 7/9

- Continue to integrate GUI with genetic algorithm
- Review and write up on project results

##### Week 10/11

- Prepare for the final report and presentation

This is the original project plan for the project. The plan for the first term is more detailed then the second as I believe due to the cast nature of the project it can be more complex to deal with certain parts as such, I made sure that for second term I didn't overdo it with tasks.

## Full project Diary

This is the first test of my diary. ~02/10/2022

# Week 26-2 Oct

This week I reviewed ideas for my project plan and added a section on risks and mitigations. I realised that creating proof of concepts and simple programs can help identify other risks and mitigations.

After talking in more depth and asking questions to my supervisor I rationalised that starting with simple constraints e.g. integers rather than real numbers or even possibly boolean would be a good start and/or first target. I have also decided that rather than starting on the gui, since this project mostly looks at the genetic algorithm rather than physical attributes it would be best to start with a simple gui and focus much more on understanding and creating the genetic algorithms.

# Week 3-9 Oct

This week I have created some sketches for a simple temporary gui which will be done in tkinter and javascript at some point of the week to help me identify which would be more suitable to program in.

After doing some research, I decided that installing sublime text to use as an IDE to construct the graphical interface would be the best option.

After some experimentation I have found that doing the project in python would be the best idea particularly for producing visual genetic algorithms.

# Week 10-16

This week I have created a simple GUI in tkinter for the project, I have done some experimentation with the features and widgets available and have an idea of how I want the GUI to look like. I have also started creating simple Genetic algorithms and am looking particularly at the TSP.

# Week 17-23

During this week I created a simple genetic algorithm to understand how they work and how it can be applied to the project. I also did some research and a write up on genetic algorithms to form the basis of my report.

# Week 24-30

For this week I have started creating the TSP genetic algorithm. From creating a simple GA from before I now have a good understanding of how genetic algorithms work and it has helped me to do this. In addition, adding a small section of encoding various problems with GA's which I will expand upon.

# Week 31-6

So far this week I have worked on the TSP genetic algorithm and am close to finishing it up. I have found that it is much better to use a class for this program rather than a series of methods. I have also updated the style and the layout of the report. Overall, I have made quite some progress this week, but I have to admit that more can be done, this was in large part due to multiple submissions and deadlines for my data analysis module which has been quite tricky. Going forward I intend to work even more on the project. I have also now included the project plan making it easier to access and work on.

# Week 7-13

This week I worked more on the report adding sections to do with genetic algorithms such as the introduction and other aspects namely being theory of coalescence, genetic drift and encoding. Also added a little bit of code.

# Week 14-20

This week I was slightly confused with the remaining part of the genetic algorithm so I decided to take a step back and do more research. Moving on from this I also worked on the report adding to the technical section as well as genetic drift and design patterns. Upon further research I have now concluded that doing the majority of the project in tkinter is not viable and requires more time than provided. So I decided I will be doing the visualisation portion of the project in pygame for now.

# Week 21-27

This week I have worked towards getting a basic TSP genetic algorithm completed, this will further reinforce my knowledge and help me with the processes that take place in evolutionary algorithms. I feel ill with covid so for some days I was not able to complete work as best as I feel I could but I am on track now. This week I have now uploaded the full TSP GA I have been working on with full comments. In the coming week I will include the visualisation for this however due to time constraints I will probably have to use a set of constant numbers.

# Week 28-2

This week I worked on the visualisation for my project in pygame and worked more on the GUI in tkinter. I also worked on the report detailing my proof of concepts and going into more detail about the encoding of my project. The test case for the TSP GA was also uploaded.

```

Term 2

# Week 23-29
Have a meeting with supervisor to discuss the project. Do research on constraint optimisation in a broad sense and how it can be further extended in the project. Research potential method for this.

# Week 30-5/ 6-12
Have been working on fixing the issues brought up such as adding documentation refactorising code and making it more understandable. Have also added a video on how to run the code and what it should display. Have been working on report so added a section on TSP but have not committed this yet as it is quite rough. My main focus is trying to expand on my project to make it better.

# Week 13-19
Created a survey of optimisation methods and compared them as well as examining their strengths and weaknesses. Improved different aspects of the report such as the introduction and the abstract.

1. Create a survey
2. Keeping working to try to expand the project I am between the idea of incorporating other CSP or solving Constraint optimisation itself.
3. Add more sections to the report.

# Week 20-26
Added a bit more to my reports and improved them a bit. Also working on different CSP problems and aim to upload them some time next week. This will help further my understanding and build my knowledge.

# Week 28-11
So far I have worked towards creating a visual for solving the knapsack problem using pygame and working on the n-queens problem with pygame. I have also been researching optimisation test functions among other things. Furthermore, I realised that the visualisation needed to be somewhat interactive and actually affect the ga so I am working on a way to implement this possibly with tkinter as it is may be a better option than doing so in pygame. Overall this has been a large 2 weeks for me I want the project to be good and thus spent hours on research alone trying to figure out the best way to approach this and feel this will benefit me in the long run. Although I must confess I did underestimate the amount of coding and learning required to complete this project though I feel this was a great opportunity to learn about this topic more in depth. In the following week I should be able to produce some of the code and updated parts of the project. Generally I have not liked uploading things as I am constantly testing things that could suit this project and uploading and then deleting things would not be a good use of my time but I do assure you I have been working hard to get this done well.

# Week 13-19
This week will be trying to finish genetic algorithm visualisations (pretty much done but need to work on a few things) this includes knapsack, TSP and n-queens, and the interactive one in tkinter. I have opted to try to do a visualisation in the interactive one with moving points on a 3d plot.

# Week 20-26
This week I have mostly been working on writing out the report and adding the necessary information. For coding I am nearly done but need to add a few things. The final assignments and events such as varsity have taken a bit of my time particularly me getting a migraine and getting sick which is why I opted to get an extension.

```

## Guide on how to run software

### GenAlgoLab Instruction Manual

This instruction manual will guide you through the steps to run and use the GenAlgoLab, a genetic algorithm project that demonstrates solutions to various constraint satisfaction problems (CSPs), including the Traveling Salesman Problem (TSP), Knapsack, and N-Queens.

#### Prerequisites

1. Python 3.6 or higher installed on your system.
2. The following Python libraries installed: Tkinter, ttkthemes
3. The source code files for the project: GenAlgoLab.py (the main GUI file), TSPGA\_application.py (TSP Genetic Algorithm application) Knapsack\_ga.py (Knapsack application) queen\_ga.py (N-Queens application), GUI.py (Interactive Genetic Algorithm application), ga.py (genetic algorithm)
4. The dependencies can be installed from requirements.txt (to do this in your terminal or command prompt use the following command: `pip install -r requirements.txt`)

#### Using the Program

In the main window, click on the "CSP" menu item in the menu bar.

1. A dropdown menu will appear with the following options: TSP, Knapsack, and Nqueen. Click on the desired option to run the corresponding genetic algorithm application.
2. TSP: Traveling Salesman Problem, Knapsack: Knapsack Problem, Nqueen: N-Queens Problem
3. Each option will open a separate window running the selected genetic algorithm application.
4. To run the Interactive Genetic Algorithm application, click on the "Interactive GA" menu item in the menu bar. A separate window will open running the interactive genetic algorithm application.
5. For more information about the GenAlgoLab project and the Traveling Salesman Problem Genetic Algorithm, click on the "Info" menu item in the menu bar. A help window will open displaying information about the project.
6. To close the main window and all open application windows, click on the "Force Quit" button in the main window.

The other programs in experiment can simple be run by in a IDLE preferably sublime or visual studio.