Final Year Project Report

Interim Submission – Final Report

Genetic Algorithm

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

December 06, 2022

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 Where other published and unpublished source materials have been used, these have been acknowledged.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

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. The approach has been tested using a series of python programs to create genetic algorithms then produce a visualisation for it.

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.

The aim of the project is to use develop an understanding of genetic algorithms and explore how they can be used to solve 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 specifically on one problem in this case being the Travelling Salesman Problem (TSP). The TSP is a problem is an example of a constraint satisfaction problem used for finding the shortest and most efficient route for a person 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. I also aim to write a report gaining insight into genetic algorithms and providing knowledge on this.

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. Key concepts on the theory of evolution will be explored like genetic drift, mutations, fitness functions and crossover functions. In the end, I plan to gain comprehensive background knowledge from the project with regards to how evolutionary driven technology can be used to help solve real-world problems.

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.

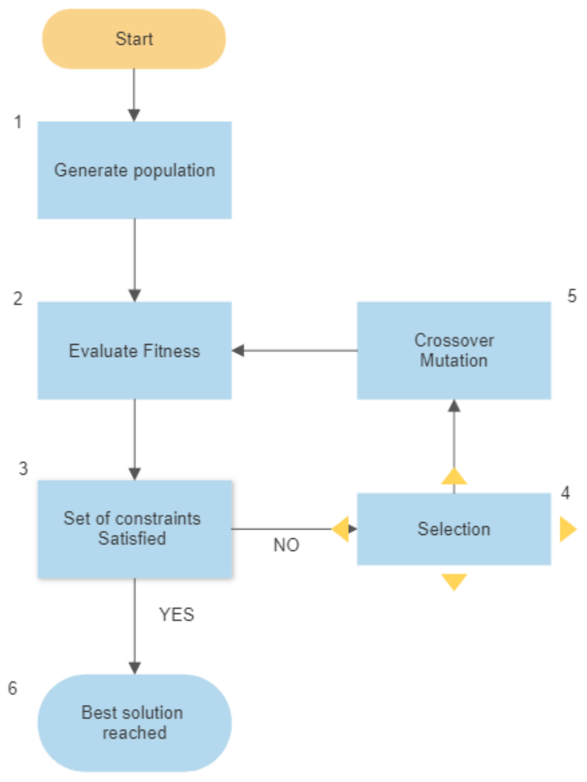


Figure 1: Flowchart of genetic algorithm

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

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 if variables X = {x_1, …, 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.

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

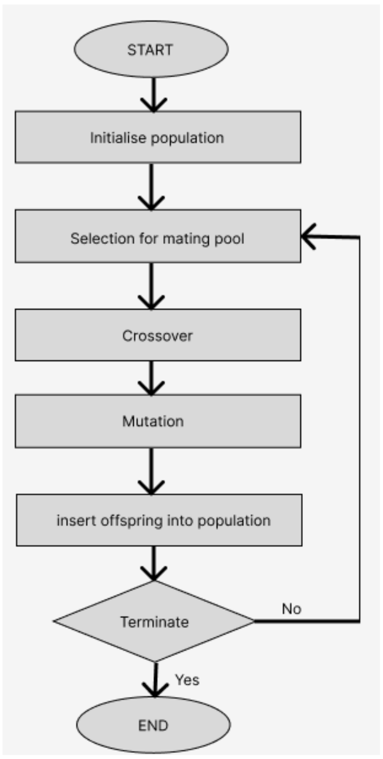


Figure 2 algorithms phases for GA

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.

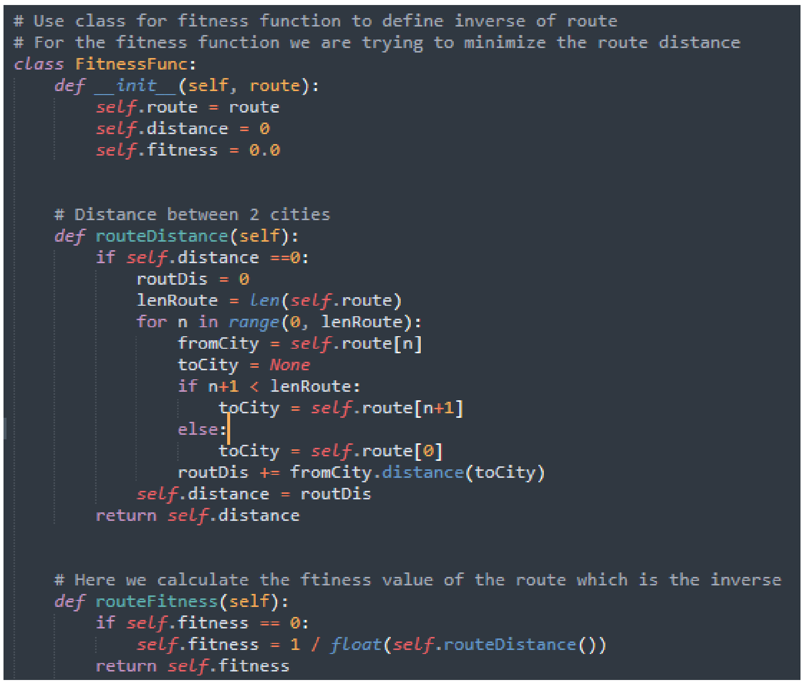
<u>Types of encoding:</u>

- 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.

<u>Fitness Function</u>

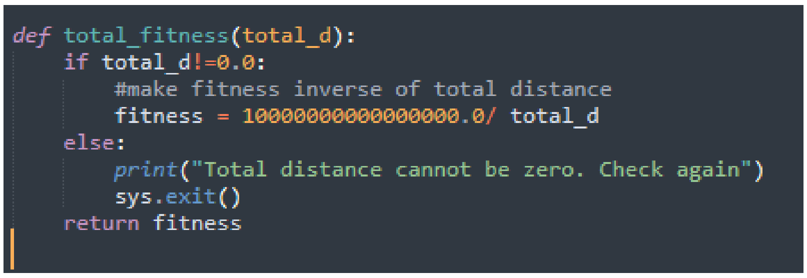
Fitness functions are used to determine how well solutions (such chromosomes) perform. The idea that fitness functions are unique to a particular issue is a key characteristic.

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 ftiness 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.

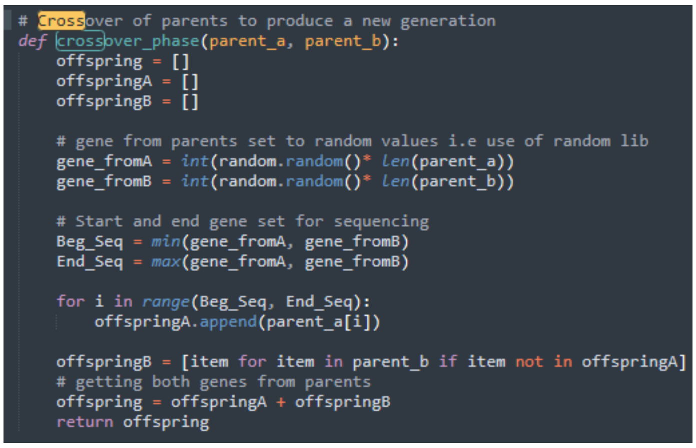


```
def total_fitness(total_d):
    if total_d!=0.0:
        #make fitness inverse of total distance
        fitness = 10000000000000000.0/ total_d
    else:
        print("Total distance cannot be zero. Check again")
        sys.exit()
    return fitness
```

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 then 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

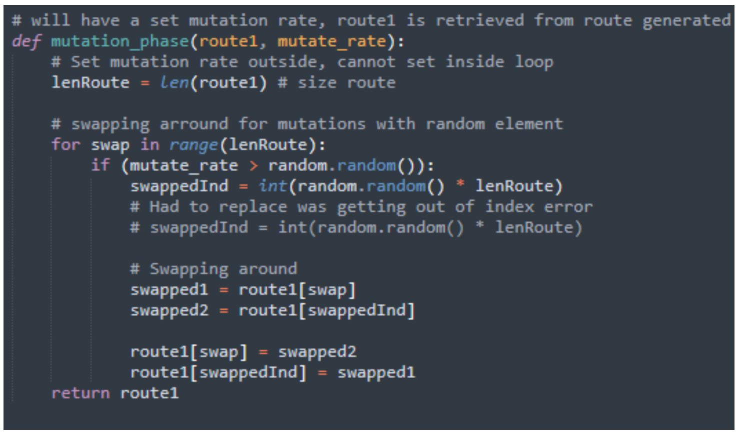


```
# Crossover of parents to produce a new generation
def crossover_phase(parent_a, parent_b):
    offspring = []
    offspringA = []
    offspringB = []

    # 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.

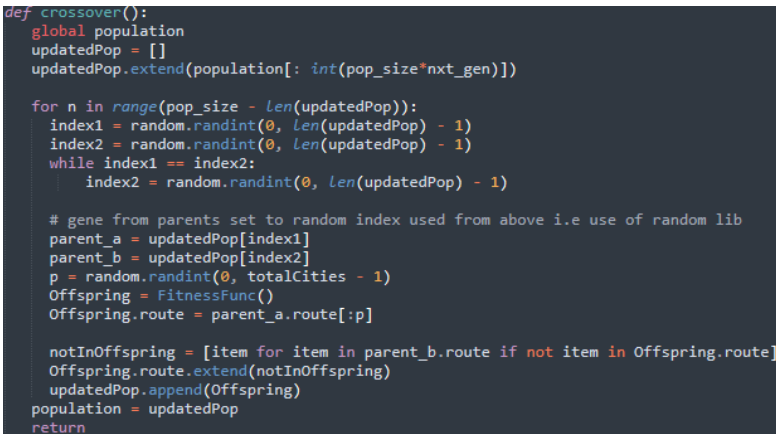
The TSP is special in that we must include every location precisely once. We can employ a unique breeding function called ordered crossover to conform to this criterion. Through ordered crossover, we first arbitrarily choose a subset of the first parent string and then populate the remaining route with 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 arround 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.

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).

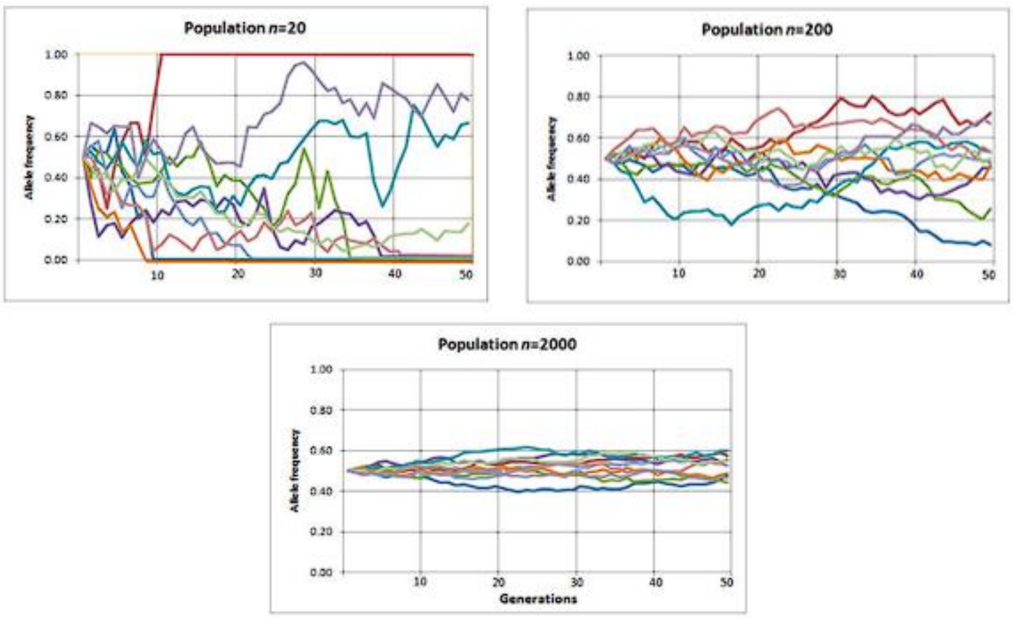


Figure 3: Random genetic drift graphs

Discuss bottleneck and founder in more detail.

<u>Population Bottleneck</u>

The Bottleneck effect is when you have some sort of major disaster or effect that in turn kills of a lot of the population resulting in a severe decrease in a populations 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 discussed be 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 to 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.

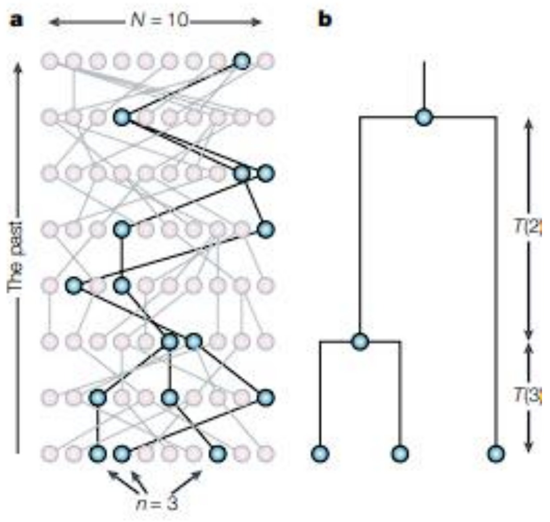


Figure 4 depicts basic principle behind coalescent

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 linages 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.

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.

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 ad 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.

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 of 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.

Design patterns

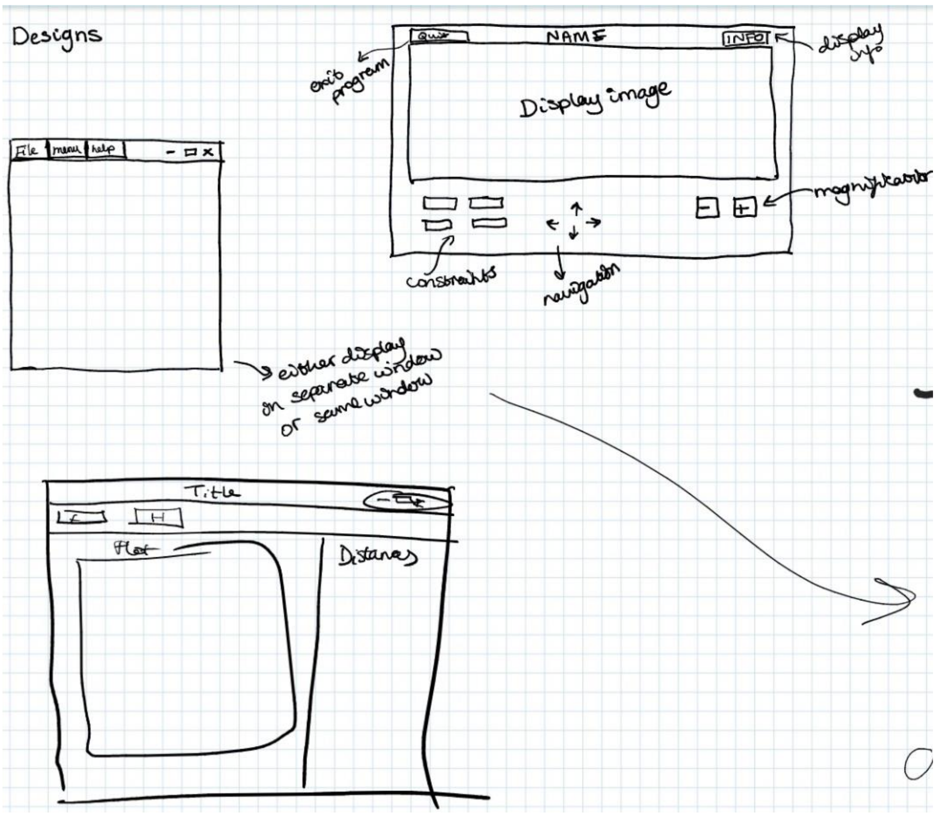
Design patterns are characterised by their use they include creational, structural, behaviour and architectural.

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

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

Design patterns are a key aspect of software engineering, although I did not use much design patterns this year, I intend to use it in a more defined way next term. After researching design patterns honestly, I couldn’t quite figure our out which one would be the most applicable. So, I decided to just use elements of design patterns to improve my application. An example of this is 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. A design pattern that I did decide to try and incorporate is the MVC design pattern. This consist of a model (pure logic), view and controller. 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.

2D Prototypes

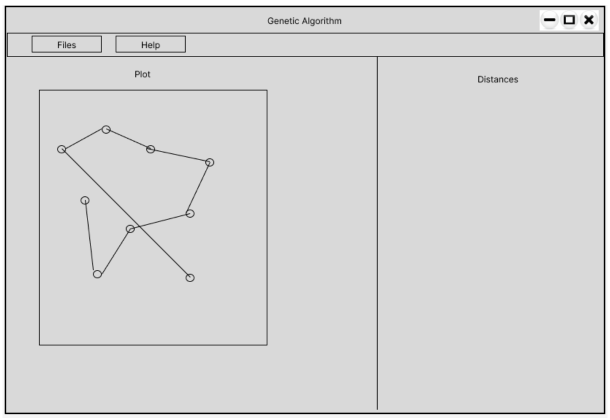


The screenshots you see here are 2D prototypes of work I did earlier in the term, I expected my application to look like these prototypes. As you can see my 2D prototypes differ from my prototype developed. It is important to show how the project developed. This demonstrates how much I learnt from working 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

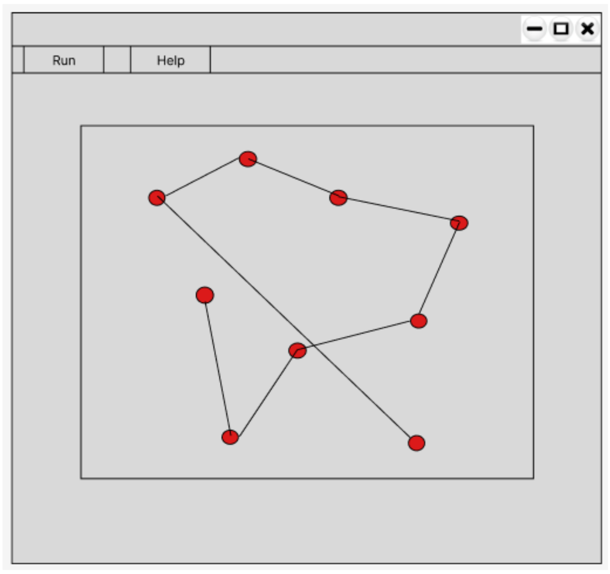
For the first implementation of the GA for constraint satisfaction, I decided to work on the travelling salesman problem. The concept of the overall project is to build a general program for constraint optimisation but for the first term I wanted to work on one constraint problem in this case TSP.

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

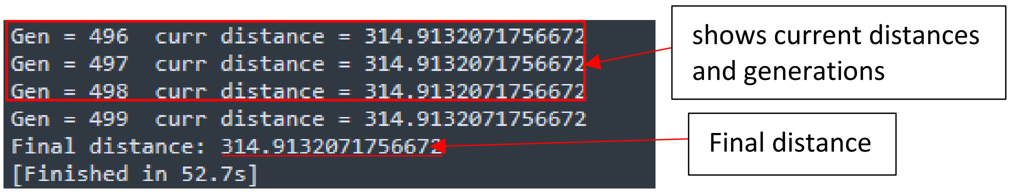


For the next prototype I decided I wanted to go a simpler route and just display the TSP on the GUI. As you can see it is still quite interactive it has buttons how help navigate. It also shows visualisation in real time allowing the user to be able to understand and truly examine how the ga works with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

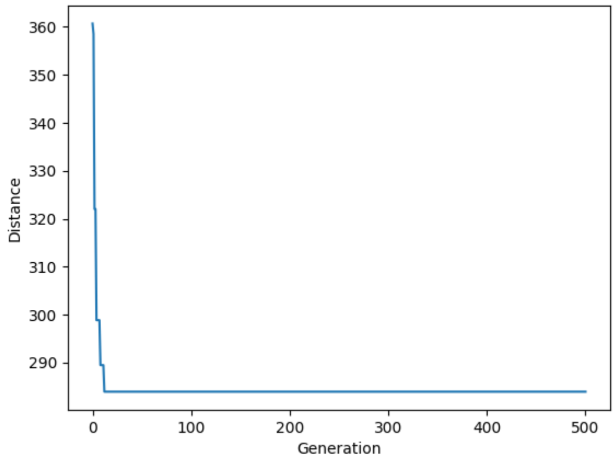
Before creating my project, I started by experimenting with genetic algorithms firstly creating a simple genetic algorithm to solve an expression problem. Following this I started working on a genetic algorithm for the travelling salesman problem in python and doing research such as plotting graphs to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

Genetic Algorithm For TSP



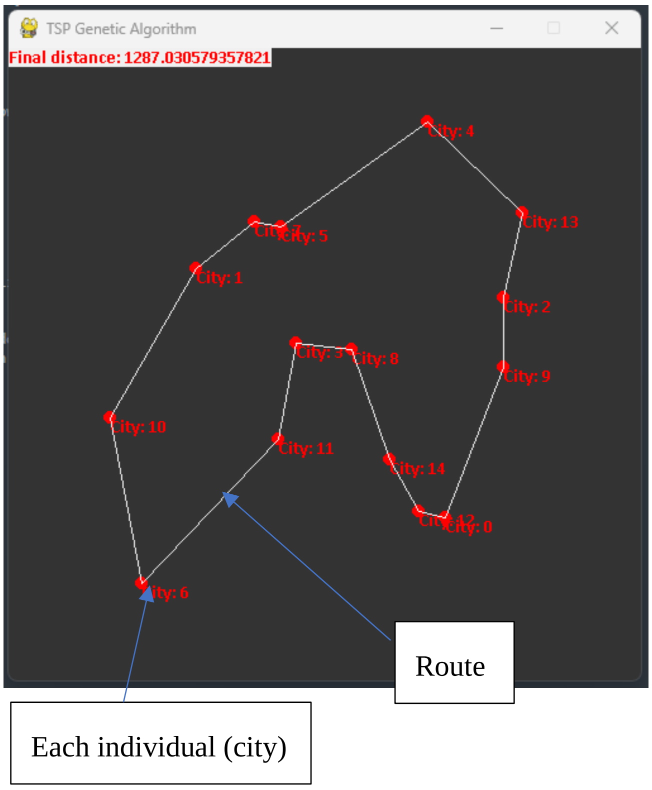
```
Gen = 496   curr distance = 314.9132071756672
Gen = 497   curr distance = 314.9132071756672
Gen = 498   curr distance = 314.9132071756672
Gen = 499   curr distance = 314.9132071756672
Final distance: 314.9132071756672
[Finished in 52.7s]
```

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



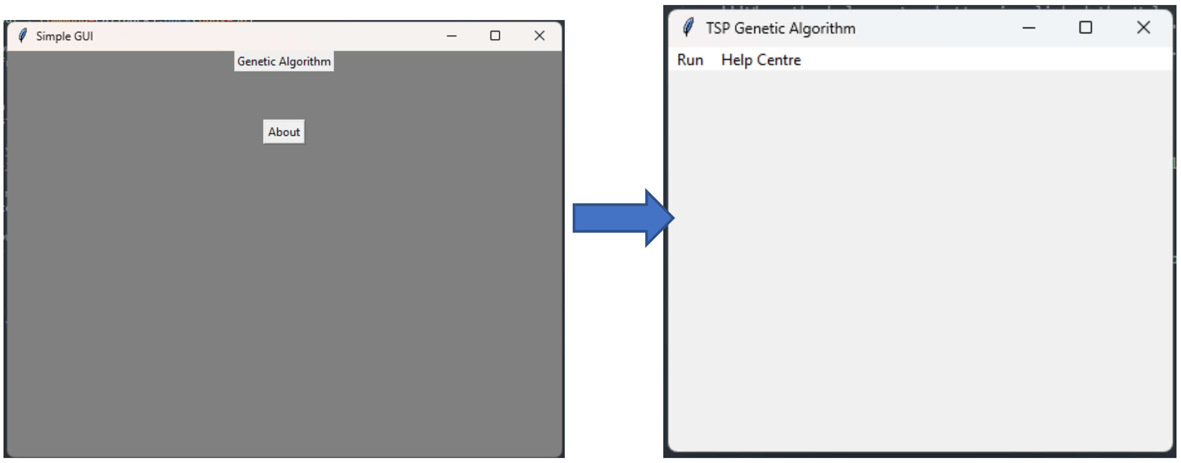
The conclusions I came to whilst doing this were that he higher the generation size the higher the time need to calculate distance and the higher the final distance is. The mutation rate also affects the distance as the higher the mutation rates the greater the final distance is. Whilst doing this I 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.

3.6 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.

3.7 Next 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

Chapter 4: **Summary of 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.

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

Bibliography

[1] En.wikipedia.org. 2022. *Genetic algorithm - Wikipedia*. [online] Available at: <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> [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.

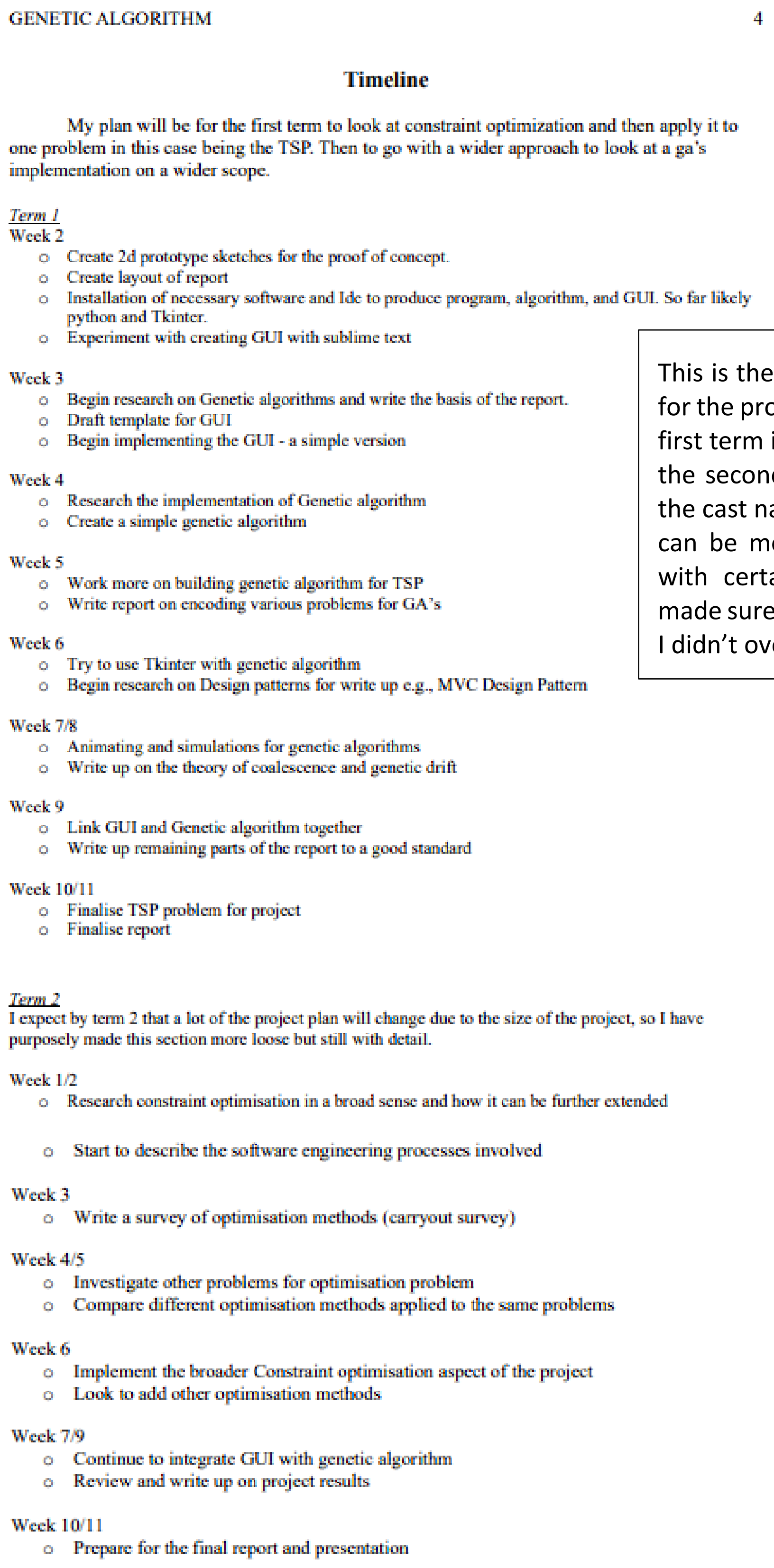
[10] Heest, M.van (no date) *Every shortcut for designers, centralized and searchable*, *Sublime Text shortcuts* | *All shortcuts for Sublime Text - Shortcuts.design*. Available at: https://shortcuts.design/tools/toolspage-sublimetext/ [Accessed: December 6, 2022].

[11] Ashlock, D. (2006) *Evolutionary computation for modeling and Optimization*. New York: Springer. [Accessed 27 November 2022]

[GD] Kryvokhyzha, D. (2018) *Genome evolution and adaptation of a successful allopolyploid, capsella Bursa-Pastoris*. dissertation. Acta Universitatis Upsaliensis [Accessed 21 November 2022].

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

Week 3
- o Begin research on Genetic algorithms and write the basis of the report.
- o Draft template for GUI
- o Begin implementing the GUI - a simple version

Week 4
- o Research the implementation of Genetic algorithm
- o Create a simple genetic algorithm

Week 5
- o Work more on building genetic algorithm for TSP
- o Write report on encoding various problems for GA's

Week 6
- o Try to use Tkinter with genetic algorithm
- o Begin research on Design patterns for write up e.g., MVC Design Pattern

Week 7/8
- o Animating and simulations for genetic algorithms
- o Write up on the theory of coalescence and genetic drift

Week 9
- o Link GUI and Genetic algorithm together
- o Write up remaining parts of the report to a good standard

Week 10/11
- o Finalise TSP problem for project
- o 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
- o Research constraint optimisation in a broad sense and how it can be further extended
- o Start to describe the software engineering processes involved

Week 3
- o Write a survey of optimisation methods (carryout survey)

Week 4/5
- o Investigate other problems for optimisation problem
- o Compare different optimisation methods applied to the same problems

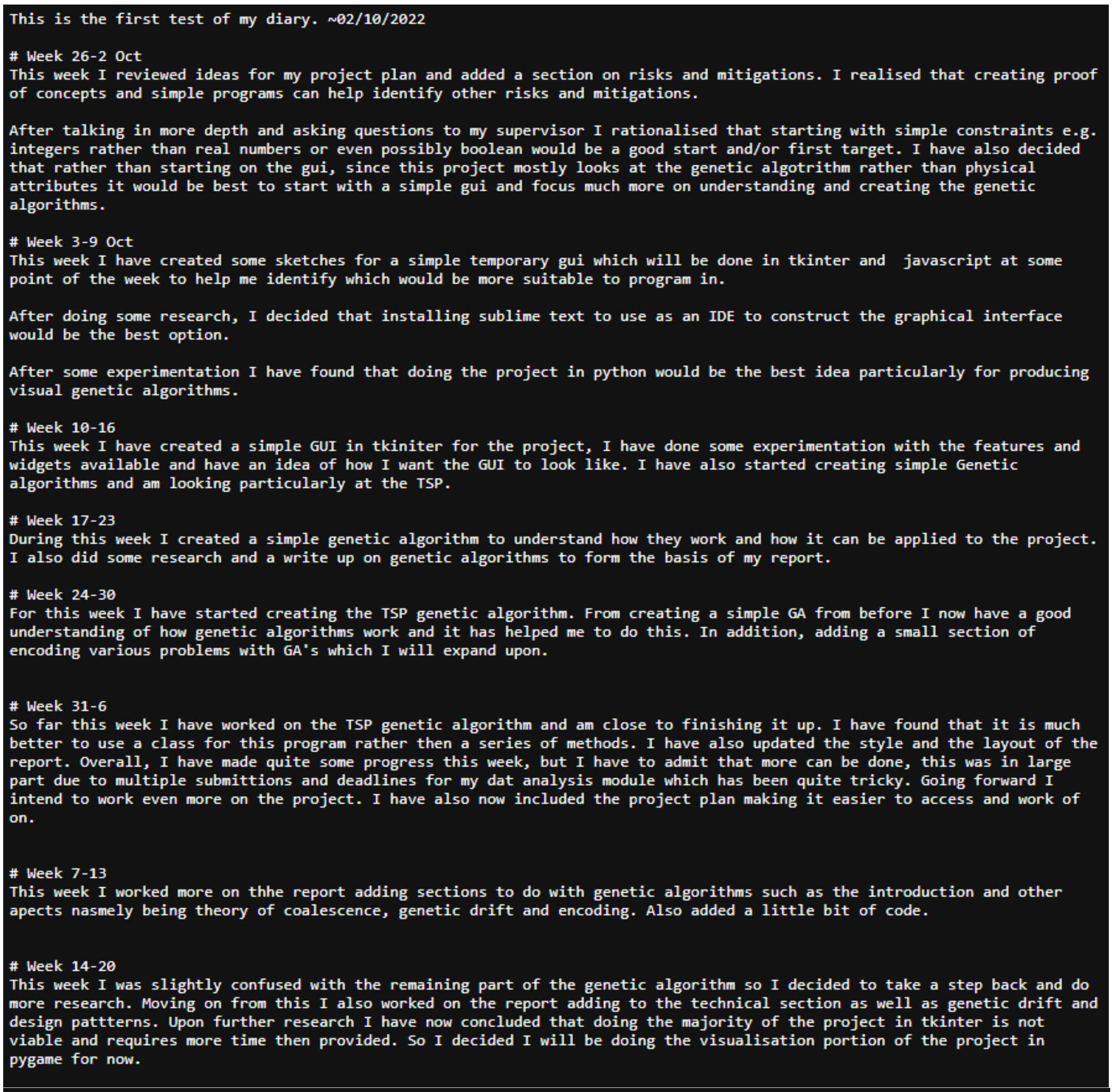
Week 6
- o Implement the broader Constraint optimisation aspect of the project
- o Look to add other optimisation methods

Week 7/9
- o Continue to integrate GUI with genetic algorithm
- o Review and write up on project results

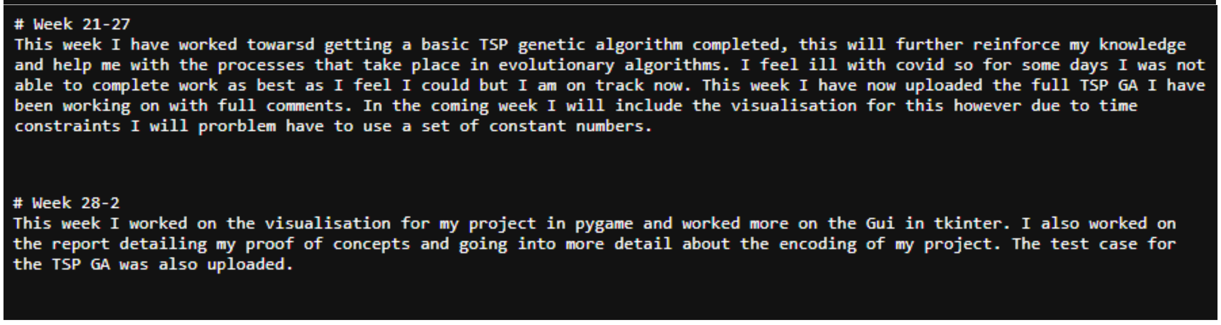
Week 10/11
- o Prepare for the final report and presentation

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 algotrithm 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 tkiniter 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 then 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 submittions and deadlines for my dat 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 of
on.


# Week 7-13
This week I worked more on thhe report adding sections to do with genetic algorithms such as the introduction and other
apects nasmely 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 pattterns. Upon further research I have now concluded that doing the majority of the project in tkinter is not
viable and requires more time then 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 towarsd 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 prorblem 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.
```