



Department of Computer Science

BSc (Hons) Computer Science

Academic Year 2024 - 2025

**Development of a Smart Sports-Specific Workout Generator
Using Optimization Algorithms**

Caspar Ashworth

2010518

A report submitted in partial fulfilment of the requirements for the degree of
Bachelor of Science

Brunel University London
Department of Computer Science
Uxbridge
Middlesex
UB8 3PH
United Kingdom
T: +44 1895 203397
F: +44 (0) 1895 251686

Abstract

This dissertation addresses the limited availability of sport-specific training for individuals from lower socioeconomic backgrounds or remote areas, and how this affects both individual development and larger population health. To solve this issue, a Python based application was developed that generates sport-specific workout plans using one of three optimization algorithms: genetic algorithm, ant colony optimization, and simulated annealing.

The problem is simplified to a weighted constraint satisfaction problem (WCSP), for which the three algorithms were implemented and integrated into an application with a GUI and database support, simulating a fully developed product. The created workout plans were evaluated based on computational efficiency and plan quality, using a large language model (LLM) to assess relevance to the sport, alignment with user skill level, and overall effectiveness.

Results show that the ant colony optimization algorithm generally produced the most effective workouts with acceptable timings. Nevertheless, all three algorithms showed comparable performance when tuned appropriately.

The findings demonstrate the feasibility of generating high-quality, sport-specific workout plans algorithmically, suggesting that such an application could be further developed and deployed as a practical, user-friendly tool.

Acknowledgements

I want to thank my supervisor Mr Philipp Bibik for all the guidance and support throughout the full process.

I certify that the work presented in the dissertation is my own unless referenced.

	Caspar Ashworth
Signature	_____
	09/04/2025
Date	_____

Total Words: 26798

Table of Contents

Abstract	i
Acknowledgements	ii
Table of Contents	iii
List of Tables.....	v
List of Figures.....	vii
1 Introduction	9
1.1 Introduction	9
1.2 Aims and Objectives.....	10
1.2 Project Approach.....	11
1.3 Dissertation Outline	12
2 Background	13
2.1 Overview.....	13
2.2 Problem.....	13
2.3 Solutions	14
2.4 Supporting Code	19
2.5 Related Work	23
3 Methodology	25
3.1 Introduction	25
3.2 Project Planning.....	25
3.3 Technologies Used	29
3.4 Ethics.....	30
4 Design	31
4.1 Introduction	31
4.2 Database Design.....	32
4.3 Algorithm Design.....	37
4.4 Fitness Design.....	41
4.5 LLM Integration Design	42
4.6 GUI Design.....	45
5 Implementation.....	48
5.1 Introduction	48
5.2 Functions	48
5.3 Main functions.....	50
5.4 GUI.....	62

5.5	Database.....	66
5.6	Story.....	66
6	Testing.....	70
6.1	Introduction	70
6.2	Unit testing.....	70
6.3	Functional testing	72
6.4	Performance evaluation	75
7	Evaluation	81
7.1	Introduction	81
7.2	Output testing/LLM-as-a-judge.....	81
7.3	Evaluation.....	86
7.4	Comment analysis	88
7.5	Errors	90
8	Conclusions.....	91
8.1	Aim	91
8.2	Objectives	91
8.3	Results of testing/evaluation	92
8.4	Future Work.....	93
	References	94
Appendix A	Personal Reflection	99
A.1	Reflection on Project.....	99
A.2	Personal Reflection.....	99
Appendix B	Ethics Documentation.....	101
B.1	Ethics Pro-forma.....	101
Appendix C	Video Demonstration	105
C.1	Link to the video	105
Appendix D	Other Appendices.....	106
	More relevant material.....	106

List of Tables

Table 1 - Conclusion of WCSP solutions.....	19
Table 2 - Technologies used for design	29
Table 3 - Technologies used for implementation.....	29
Table 4 - Technologies used for testing.....	30
Table 5 - Technologies used for supporting activities	30
Table 6 - Example workout plan	32
Table 7 - Sports table design.....	33
Table 8 - Exercise table design.....	34
Table 9 - People table design.....	35
Table 10 - Testing table design	36
Table 11 - Fitness function objectives	42
Table 12 - User feedback prompt design	44
Table 13 - Evaluation prompt design.....	45
Table 14 - Functions.....	50
Table 15 - Fitness function	51
Table 16 - llmConstraints function.....	53
Table 17 - startingPop function.....	54
Table 18 - fitnessCalc function.....	54
Table 19 - selection function.....	55
Table 20 - crossover function.....	55
Table 21 - mutation function	56
Table 22 - heuristic function.....	58
Table 23 - SelectExercises function (Stretches)	58
Table 24 - SelectExercises function (Exercises)	59
Table 25 - updatePheromones function.....	59
Table 26 - restDays function.....	60
Table 27 - Input validation checks	64
Table 28 - Unit test results.....	71
Table 29 - Functional testing input.....	72
Table 30 - Profiling results	77
Table 31 - Profiling results with feedback	78
Table 32 - Example workouts from internet.....	81
Table 33 - LLM-as-a-judge results (pre-processing)	83
Table 34 - LLM-as-a-judge results (post-processing)	85

Table 35 - Edge case 1 user	87
Table 36 - Edge case 1 results	87
Table 37 - Edge case 2 user	87
Table 38 - Edge case 2 results	87
Table 39 - Comment analysis results	88
Table 40 - Conclusion objectives.....	91

List of Figures

Figure 1 – A flowchart showing the outline of my dissertation	12
Figure 2 - Picture showing how ant colony optimization.....	18
Figure 3 - Route selection algorithm for ACO	18
Figure 4 – A section of my list used for planning	26
Figure 5 – Showing my project timeline	27
Figure 6 – A screenshot of a list from my phone	28
Figure 7 – Flowchart showing the flow of my program	32
Figure 8 – Flowchart for genetic algorithms.....	37
Figure 9 – Flowchart for simulated annealing.....	39
Figure 10 – Flowchart for ant colony optimization.....	40
Figure 11 – Wireframe of my GUI	46
Figure 12 – Wireframe of my GUI with user feedback.....	47
Figure 13 – Flowchart of the fitness function	50
Figure 14 – Flowchart of my genetic algorithm.....	53
Figure 15 – Flowchart of my ant colony optimization algorithm.....	57
Figure 16 – Flowchart of my simulated annealing algorithm	60
Figure 17 - Picture of simulated annealing code	61
Figure 18 – Flowchart of my GUI flow	62
Figure 19 – Screenshot of my GUI.....	63
Figure 20 – Screenshot of an error message from my program	63
Figure 21 – Screenshot of my GUI with feedback box.....	65
Figure 22 - Screenshot of a workout in my GUI.....	65
Figure 23 - Picture of duplicate exercise fix from fitness function	67
Figure 24 - Picture of constraint exercise part of fitness function.....	68
Figure 25 – Screenshot of a workout plan from my program before user feedback (GA)	73
Figure 26 - Screenshot of a workout plan from my program after user feedback (GA)	73
Figure 27 - Screenshot of a workout plan from my program before user feedback (SA)	74
Figure 28 - Screenshot of a workout plan from my program after user feedback (SA)	74
Figure 29 - Screenshot of a workout plan from my program before user feedback (ACO).....	75
Figure 30 - Screenshot of a workout plan from my program after user feedback (ACO)	75
Figure 31 – Performance evaluation inputs for my GA.....	76
Figure 32 - Performance evaluation inputs for my SA.....	76
Figure 33 - Performance evaluation inputs for my ACO	76
Figure 34 - Graph showing measured vs predicted time complexity.....	79

Figure 35 – Screenshot of ChatGPT generated code for creating random user information	82
Figure 36 – Screenshot of the context part of my prompt.....	82
Figure 37 - Screenshot of the instruction part of my prompt.....	83
Figure 38 - Screenshot of the input data part of my prompt.....	83
Figure 39 - Screenshot of the output format part of my prompt.....	83
Figure 40 – Stacked bar chart of LLM-as-a-judge data pre-processing	84
Figure 41 – Scatter graph of LLM-as-a-judge data pre-processing.....	84
Figure 42 – Stacked bar chart of LLM-as-a-judge data post-processing.....	85
Figure 43 – Scatter graph of LLM-as-a-judge data post-processing.....	86

1 Introduction

1.1 Introduction

The Problem

Within the world of sport, sport-specific training plans are often essential for skill development, physical, and mental conditioning (Granacher, U. and Borde, R, 2017).

However, access to these specialized training is limited and often expensive, creating barriers for athletes, particularly those from low-income backgrounds or remote areas. Between November of 2022 - 2023 Sport England found that only 52.6% of people in lower socioeconomic groups (NS SEC 6-8) did over 150 minutes of activity a week, compared to 72.7% of NS SEC 1-2 (sportengland.org, 2020).

Causes

- Lack of experienced coaches: Specialized, high-quality coaching is vital for athletes aspiring to excel, yet there are not enough qualified coaches available across sports disciplines, particularly in smaller communities or emerging sports. In a study by UK Coaching they found that only 50% of the 6373 coaches who participated had any formal training (UKCoaching, 2020). The demand for good coaching and the lack of availability often pushes the price even higher.
- Cost of personalized training: Personalized training typically requires significant investments in time and money. Professional coaches who specialize in personal training are often associated with higher fees due to the extensive planning and attention required for individual athletic needs. For example according to The Fitness Group (thefitnessgrp.co.uk, 2023) the average cost of a personal trainer (Not sport specific) in London is between £50 - £150. As a result, only athletes with the financial means or access to funding can afford high-level sport-specific coaching.

Impact

The lack of affordable, tailored training disproportionately affects athletes who already face socioeconomic challenges. These limitations mean that individuals from lower-income backgrounds or more remote areas are unable to benefit from the same opportunities as other athletes from better backgrounds or areas. The inability to access sport-specific training can slow progression, reducing opportunities for these athletes to pursue higher-level competitive play or professional careers. It may also discourage people from participating in any sport at all which is known to be responsible for not only many mental

health issues (Mental Health Foundation, 2025) but also potential for crime and antisocial behaviour (Nichols, G, 2014).

Why is it a Problem

The inaccessibility of sport-specific training creates inequities in athletic development, limiting opportunities for individuals to achieve their full potential in sports. This can have broader implications, affecting both physical and mental health. Access to sports is associated with better mental health, such as better self-esteem, reduced stress, and improved social skills (sportengland.org, 2024). Physically, participation in structured, well-coached sport promotes better overall well-being and reduces the risk of chronic illnesses. Limited access not only stops personal and athletic growth but also deprives athletes who might otherwise benefit from and contribute to the sports community.

1.2 Aims and Objectives

Aim

The aim of my project is to create a program which will make a sport-specific workout plan (Specifically a gym plan/routine). It should do this by taking in information about the user such as skill level and the chosen sport, this will then be used to create a plan that targets the specific skills and muscle groups that are important in the sport and fills the time effectively. The program should also be able to take in comments/suggestions from the user to prioritise or avoid specific muscle groups or exercises due to injury or preference.

Objectives

1. Conduct background research on similar products to discover how they work, what they do well, and what I should avoid.
2. Simplify the main workout creation function to a mathematical problem or set of problems.
3. Research ways this has been solved in other academic papers as to choose the 'best' methods to test against each other.
4. Investigate the use of popular large-language-models that could be used for the user comments/suggestion's function.
5. Plan out my project at a high-level, specifically my methodology and the technologies I will be using.
6. Plan out implementation at a high level, what functions I will be using, the data and tables that I will need to make and how it moves through the system, and how I will be evaluating the program.

7. Test the program using standard unit testing and error handling tests.
8. Evaluate using statistical tests and the computational complexity of my algorithms to find out if I have been successful in achieving my aim.

1.2 Project Approach

Introduction

As you have seen this will be an overview of the project as to provide an understanding of what the project is, why it's important, and how the dissertation will be structured.

Background

This section covers why the problem is real, how I plan to solve it, other solutions that exist, why my solution is appropriate using other scientific studies and data and should provide all the necessary information for the reader to understand the rest of the dissertation.

Methodology

Here I go over how I planned my project, what I will be using to keep myself on track throughout the project, and why I chose these techniques.

Design

In this section I show how I designed my program including, the algorithm choice and design, the data flow and table design, and the front-end design. I also cover the technology I used and the reason I chose it.

Implementation

This section covers both how my program works in detail and the story of implementation, where I talk about some of the challenges I faced and how I solved them.

Testing

This section covers what techniques I used to test my code, why I used them, and the results of testing.

Evaluation

Finally, the evaluation will go over the results of my testing in more detail as to show if and how I fulfilled my aim, as well as what changes I would make if I were to do it again or work on it further.

1.3 Dissertation Outline

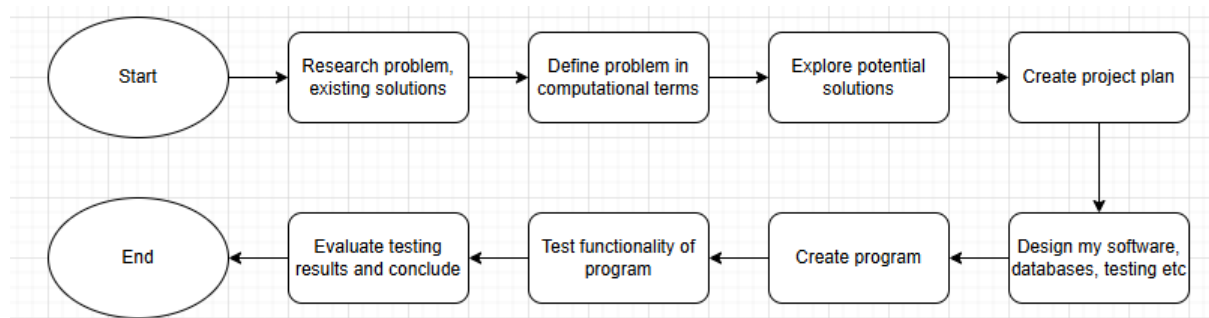


Figure 1 – A flowchart showing the outline of my dissertation

Above is a flow chart showing how I went about my project. As you can see the first three sections can be combined to my background section, this is where I aim to meet learning objective 1 and 2, as well as my personal objectives 1, 2, 3, and 4.

I then create my project plan, including how I structure my project, time estimates, and testing plans as well as the ethics. This should cover learning objectives 3, 5, and 7, as well as my personal objectives 5 and 6.

Next, I design my functions, database, GUI, and prompts using a variety of methods and inspiration from my background research. This covers learning objectives 2 and 3, and personal objective 6.

Then is the creation of the program, this is covered in the implementation chapter and will communicate how it works, broken down into specific functions/algorithms, why I did it this way, and what problems I faced. This covers learning objectives 3 and 6.

Testing the functionality of the program comes after, here I use standard software tests to ensure my program works as intended. This will cover learning objective 4, and personal objective 7.

Finally, I evaluate the testing results and conclude my dissertation. This will include details on what worked, what could have been done differently, and what I would do if I had more time. It will cover learning objectives 4, 5, and 6, and personal objective 8.

2 Background

2.1 Overview

In my background I discuss the problem in more detail, including proof that it exists and the problem abstraction. I then introduce mathematical optimization and more specifically constraint satisfaction problems (CSP), I talk about different methods that can be used to solve CSP's. I then review other solutions to the problem that currently exists. Finally, I cover llm-as-a-judge and my evaluation techniques. I hope by the end of this chapter the problem, solutions, and evaluation will be clear and my choices in future parts of the dissertation will make sense to you, the reader.

2.2 Problem

Proof of Problem

As presented in the problem definition section I have used statistics from Sport England that show in lower socioeconomic groups only 52.6% of people participate in over 150 minutes of exercise a week (sportengland.org, 2020) compared to 72.7% of people in higher socioeconomic groups. I also used statistics gathered by YouGov for UK Coaching which identified only 50% of the 6373 coaches asked have official qualifications (UKCoaching, 2020), that includes coaches of all sports so I believe given a less popular sport the number of coaches would be even smaller, this can be supported by the numbers reported by British American Football in the year 2020-2021 where they have 582 full contact coaches (British American Football, 2021) of the estimated 3,106,053 coaches in the UK (UKCoaching, 2020). From this you can see the potential difficulty in finding a qualified coach for a lesser-known sport.

Problem abstraction

The main problem in creating a personalized, sport-specific workout plan involves selecting the best set of exercises that match a person's available time, chosen sport, skill level, and personal feedback. Each potential exercise can have different levels of relevance and suitability based on these factors, effectively assigning each exercise a distinct value or "weight". Additionally, some exercises may violate constraints set by the user's preferences or physical limitations.

This scenario fits a weighted constraint satisfaction problem (WCSP). Unlike constraint satisfaction problems (CSPs), which focus on finding solutions that perfectly satisfy all constraints, WCSPs can allow for some constraints to be violated to some degree. Each constraint violation carries a penalty or cost, this means it is now trying to identify a solution

that minimizes the overall weighted penalty, effectively balancing constraint satisfaction and solution quality.

2.3 Solutions

Mathematical optimization

Mathematical optimization is a branch of mathematics that deals with finding the best solution to a problem with multiple possible solutions (Floudas and Pardalos, 2009). It is used across many fields such as economics, engineering, and computer science to make decisions, allocate resources, and solve problems.

Optimization problems consist of:

- The objective function ($f(x)$) is the output you are trying to minimize or maximize.
- Variables (x_1, x_2, x_3) are the inputs that can be adjusted to find the best outcome.
- Constraints (Ω) are the conditions or restrictions that place limits on the variables.

The two main categories of optimization problems are based on whether the variables are discrete or continuous. In a discrete optimization problem, an object (such as an integer or graph) must be found from a countable set ("In mathematics, a set is countable if either it is finite or it can be made in one-to-one correspondence with the set of natural numbers" (SISL, n.d)). A problem with continuous variables or "continuous optimization" is where arguments from a continuous set must be found.

Optimization problems are defined by the objective function, variables, and constraints, some different types of optimization problems are:

- Linear optimization, where the objective function and constraints are linear.
- Nonlinear optimization, where the objective function and constraints are non-linear.
- Integer optimization, where the decision variables are limited to integers.
- Convex optimization, where the objective function is convex, or concave (Minimization and maximization respectively) and the constraint set is convex.
- Stochastic optimization, where the data or constraints are uncertain and therefore requires probabilistic models.
-

Constraint satisfaction problem

Constraint satisfaction problems (CSP) can be seen as a subset of mathematical optimization, this is because they are also trying to find a solution that satisfies a set of constraints imposed on a set of variables (Christophe Lecoutre, 2013). However, CSP's are only focused on feasibility and not optimization, this makes the answer useful for solving a problem that satisfies all constraints and therefore doesn't require any trade-offs. Another difference is that CSP's may not be able to be answered where mathematical optimization will always have answers, they just may not be very 'good'. The general goal of solving a CSP is to find an assignment of values to all variables so that every variable is assigned a value from its domain and all the constraints are satisfied.

A CSP consists of three components:

- Variables, these are unknowns that are assigned values.
- Domains, the set of possible values a variable could be.
- Constraints, rules or restrictions that specify relationships between variables and limit the combinations of values they can simultaneously take.

Usually, CSPs with a finite domain are solved using a type of search, for example backtracking. Backtracking is a recursive algorithm that assigns a value to each variable checking if any constraints are violated, if any are it backtracks and tries a new value.

Another search technique is constraint propagation (IBM, 2024), this is where constraints are used to limit the possible values of a variable, which in turn creates new constraints for other variables. For example, if a variable can take values from 1 to 10 and must be even, the domain can be reduced to 2, 4, 6, 8, and 10.

A third type of search technique is local search. In this technique the algorithm iteratively makes changes to improve constraint satisfaction. This technique can fail to find a solution even if one is possible. (Schiex et al., 1998)

Weighted constraint satisfaction problem

Weighted constraint satisfaction problems (WCSP) are a type of constraint satisfaction problem where constraints can be violated (Stanislav Žitný 2012). This is useful for problems which don't have a perfect solution or if we want to find a minimal cost solution, therefore it can be used in many real-world situations (SMU, 2006). The goal of the solution is to find a solution which violates the least constraints.

The components are the same as in a normal constraint satisfaction problem however constraints are now associated with a weight/cost that quantifies the penalty for violating the constraint.

Branch and Bound

One technique for solving WCSP's is the branch and bound method, this is where the possible assignment of values to variables is set up in a tree structure where each node in the tree corresponds to a partial assignment, and branches represent decisions for assigning values to the next variable. At each node the branch with the lower cost is calculated, if this is worse than the best-known solution then it's pruned. Variables are incrementally assigned values; these are shown as nodes that represent the updated state of the solution. To explore the tree methods such as breadth-first search or depth-first search are often used (Baeldung, 2024).

Local Search – Hill climbing

Another method is local search. This is a heuristic method that can be used in lots of different ways, it is often used for complex problems where exact methods are not possible or too computationally difficult. One type of local search is hill-climbing, in this method it starts with a solution (potentially random), assess neighbouring solutions and transitions to the neighbour with the highest objective function if it's an improvement on the current solution, this continues until no better neighbouring solutions exist. How the neighbour is chosen differs depending on the type of hill climb that is being used, for example simple hill-climbing chooses the first neighbour that improves on it, while steepest-ascent hill-climbing evaluates all the neighbours before choosing the best option. This method is easy to implement and works best on small search spaces. However, it can get stuck in local optima easily and has limited exploration of the search space (GeeksforGeeks, 2024a).

Local Search – Simulated Annealing

Another type of local search is simulated annealing. This method is inspired by the annealing process in metallurgy in the sense that it allows for moves to worse solutions to escape local optima, with the probability of these moves decreasing over time. The process starts with a solution and a temperature, it then moves to a neighbouring solution with a certain probability. While doing this a cooling schedule lowers the temperature, the likelihood of selecting worse solutions decreases with the temperature. This method can escape local optima and is therefore better at exploring the space. However, it requires tuning of the temperature and cooling schedule, and it can also be computationally expensive.

Local Search – Genetic Algorithms

Another method for local search is genetic algorithms (GA), these were developed by Holland and his colleagues in the 1960s and 1970s and are based on evolution theory. In GA terminology a solution vector is called a chromosome, these are made of genes which are discrete units and control one feature of the chromosome. The variable used to measure the success of a solution is called fitness. GA's operate on several chromosomes called a population, these are normally randomly initialized, the search results in the population becoming fitter and fitter until a single solution dominates. GA's have two operators that create new solutions, these are mutation and crossover. In crossover two chromosomes (parents) are combined to form a new chromosome (offspring), the parents are picked from the population with preference towards fitness as to hopefully create offspring with good genes. The mutation operator introduces random change into the chromosomes generally at the gene level, generally the rate of mutation is relatively small, it is there to reintroduce genetic diversity as crossover leads the population to converge, this helps escape local optima. Finally, reproduction is the process of selecting chromosomes for the next generation based on fitness. (Ramezani & Babamir, 2005).

Ant Colony Optimization

Leaving local search, another method for solving WCSP is ant colony optimization (ACO). This is a metaheuristic search method based on how ants use pheromones in real life, it was first proposed by M. Dorigo in his PhD thesis (Dorigo, 1992) which was published in 1992. It imitates how ants find food, in the picture below (GeeksforGeeks, 2024b) you can see there are two paths from the colony to the food, the ants follow both paths but the ants on the shorter path get to the food first, due to the longer path not being completed yet there are no pheromones connecting it so the ants are more likely to take the shorter path back. Now the concentration of the pheromones on the shorter path are much higher due to it being walked more so ants are more likely to keep using it over the long path.

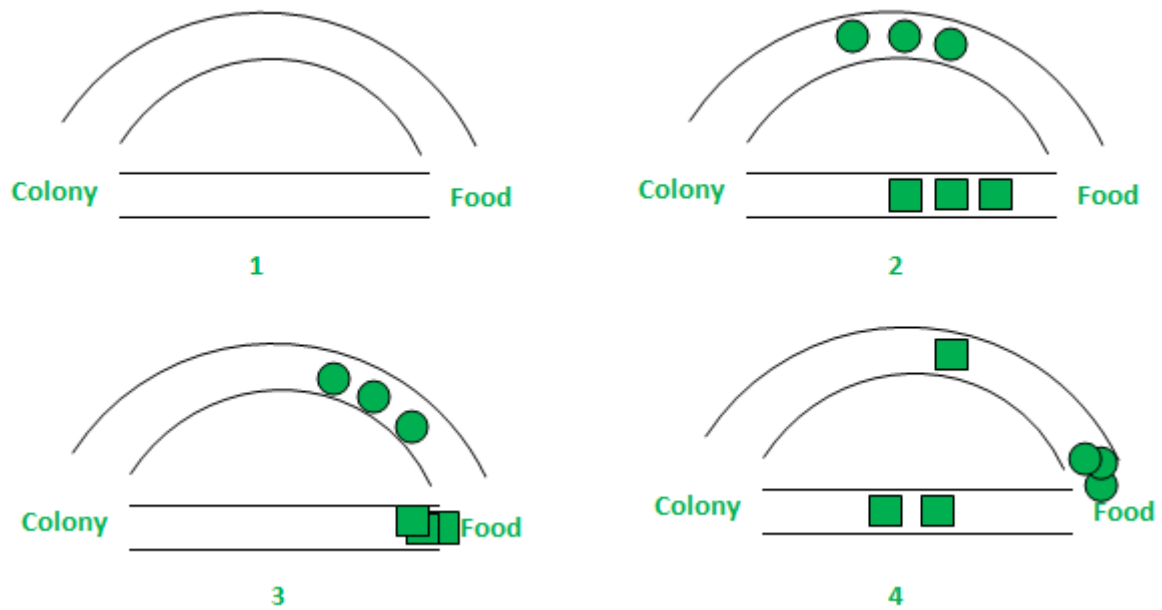


Figure 2 - Picture showing how ant colony optimization

In order to translate this into an algorithm, we can use three rules, route selection, pheromone update, and evaporation. For route selection we use the algorithm below.

$$p_{ij}^k \propto \frac{\text{Pheromone from node } i \text{ to } j}{\text{Sum of Pheromone for all valid paths}}$$

Figure 3 - Route selection algorithm for ACO

(Gambardella & Dorigo, 2007)

Conclusion

Technique	Strengths	Weaknesses
Branch and bound	<ul style="list-style-type: none"> - Guarantees optimal solutions. - Clear structured approach. 	<ul style="list-style-type: none"> - High computational cost. - Less effective for large solution spaces.
Hill Climbing	<ul style="list-style-type: none"> - Simple implementation. - Efficient for small search spaces. 	<ul style="list-style-type: none"> - Easily trapped in local optima. - Limited global exploration.
Simulated Annealing	<ul style="list-style-type: none"> - Better at avoiding local optima. - Flexible and effective in complex landscapes. 	<ul style="list-style-type: none"> - Complex parameter tuning (cooling schedule, initial temperature).

		- Computationally intensive for large-scale problems.
Genetic Algorithm	<ul style="list-style-type: none"> - Excellent global exploration capability. - Robust to complex constraints. - Scalable for large problems. 	<ul style="list-style-type: none"> - Computationally expensive. - Complex parameter tuning (mutation rate, crossover method).
Ant Colony Optimization	<ul style="list-style-type: none"> - Effective at dynamically adapting to changing constraints. 	<ul style="list-style-type: none"> - Complex parameter tuning (pheromone evaporation rate, number of ants). - Higher computational complexity.

Table 1 - Conclusion of WCSP solutions

2.4 Supporting Code

Fitness Function

For many of these techniques a good fitness function is required to guide the algorithm. This is especially important with genetic algorithms as they rely on the fitness to choose the next solution by quantifying how well the result fulfils the objectives and constraints. Usually, the fitness function will translate the objectives/constraints into a number which it aims to optimize. For CSPs and specifically WCSPs fitness functions usually use a penalty-based approach (Michalewicz & Schoenauer, 1996), this is where the fitness is assessed on each constraint and penalized if it violates any.

Penalty-Based Fitness

As described penalty-based fitness functions are very common in CSPs, this is because they can be made relatively easily by assigning penalties to each constraint. The penalties should be relative to the constraints for example a more important constraint should cause a larger penalty, finding the correct balance for this can be difficult. Another difficulty with this method is the program getting stuck in local optima due to the program not being able to step back at all, this can be solved in other parts of the algorithm such as mutation and crossover in genetic algorithms or the temperature function in simulated annealing.

Dynamic Penalty-Based Fitness

One way of solving the problem of getting stuck in local optima from penalty-based fitness functions within the function itself is by using a dynamic fitness function. This method involves penalties that change based on factors such as the iteration it's on much like

simulated annealing. This has shown improved results in complex, constrained optimization problems by balancing exploration and exploitation phases effectively (Chen et al., 2009).

Weighted Sum Fitness

In programs involving multiple objectives or complex weighted constraints, a weighted sum approach can be used. This method consolidates multiple objectives into a single fitness measure by assigning weights to each component (Zimmermann, 1984). The effectiveness of this strategy depends on accurately picking these weights, reflecting the true priorities of the problem.

Fitness Conclusion

In summary, the selection and design of a fitness function significantly influences the performance of some genetic algorithms in solving CSPs and WCSPs. Studies have shown that a hybrid approach of combining dynamic penalties with adaptive weighting often yields better results in complex scenarios (Wilkerson & Tauritz, 2010). Good fitness function design requires careful consideration of the problem as to properly assign weighting and importance of each constraint.

LLM-as-a-judge

Large language models (LLMs) are a type of advanced artificial intelligence (AI) which are made for natural language processing. This is a subfield of AI concerned with allowing computers to process data in the form of natural language. They are trained on massive amounts of data and are built on machine learning, specifically a type of neural network called a transformer model. Recently they have gained popularity, specifically with the release of ChatGPT by OpenAI in 2022, this was the first consumer facing product made by them and therefore allowed for many less-technical people to access and use it. Since then it has only gained popularity in many different fields of research as well as use in products/software made for customer use. ChatGPT is however not the only popular LLM, companies such as Google and Meta have created and pushed their own LLMs (Gemini and Llama respectively), they have benefited from the existing control over the software/hardware industry to push models into use and their massive capital to train powerful models quickly.

LLM-as-a-Judge is a new concept which explores using an LLM as an evaluation tool. It allows for large scale and automated evaluation of data, however, is a new idea and therefore isn't

always immediately respected. It involves choosing a relevant model and designing a prompt which reduces as much bias as possible while keeping it accurate.

One of the main advantages of using LLM-as-a-judge is the scalability and efficiency. Unlike humans, an LLM can complete extensive evaluations in a row without dealing with fatigue or human bias. In comparison to human experts, they are also considerably cheaper on a large scale. Another advantage is consistency, as LLMs are trained on such large amounts of data that they are often much more consistent than a group of humans who have each been trained on different course, had different life experiences, and can feel differently which may affect their evaluations. (Huang et al., 2024)

However, a disadvantage to using LLM-as-a-judge is the potential for bias, as they are trained on so much data they may inherit bias which could show in the evaluation, (Bender et al., 2021) brings up the point that due to these models using the internet to train they will have been trained on predominantly data posted by younger people from more developed countries. Another issue with LLM-as-a-judge is the reliance on prompt engineering, as a bad prompt can lead to unreliable or inconsistent results as shown in (Agrawal et al., 2023), and as the field of prompt engineering grows with the use of LLMs it is becoming more complicated and more specialized.

Recent academic research has demonstrated practical applications and effectiveness of LLM-based evaluations. For instance, (Ormerod et al., 2021) showed that GPT-based models could effectively grade essays, providing detailed feedback comparable to humans. I understand that there can be issues using LLM-as-a-judge. However, I believe with sufficient prompt design and testing I can implement it successfully and gain interesting and accurate results on my programs output.

Prompt Engineering

“A prompt is a set of instructions provided to an LLM that programs the LLM by customizing it and/or enhancing or refining its capabilities” (Mixpaper, 2023)

Designing a prompt is known as prompt engineering, this is the process of designing and optimizing prompts (inputs) to elicit desired outputs from generative AI models and could be conceived as “programming in natural language” (Reynolds & McDonell, 2021). Arguably this is the most important part of using LLM-as-a-Judge as its the main part one can control.

This paper (Springer, 2023) specifies 4 main parts to a prompt: instruction, context, input data, and output indicator. The instruction is the task or instruction you want the LLM to complete, the context is any other background information that may help the LLM understand the prompt, the input data is the input or question we want the LLM to answer, and finally the output indicator specifies the type or format of the data.

OpenAI have provided six strategies to help create a good prompt (OpenAI, 2024). These are:

- Write clear instruction.
- Provide reference text.
- Split complex tasks into simpler subtasks.
- Give GPTs time to think.
- Use external tools.
- Test changes systematically.

These alongside a good framework can help create a clear and accurate prompt.

One framework is the CLEAR framework (Springer, 2023).

This can be broken down into:

- Concise (“A concise prompt removes superfluous information, allowing AI language models to focus on the most important aspects of the task”)
- Logical (“A logically structured prompt enables AI models to better comprehend the context and relationships between various concepts”)
- Explicit (“Explicit prompts provide precise instructions regarding the desired output format, content, or scope, thereby reducing the likelihood of receiving unanticipated or irrelevant responses from the AI model”)
- Adaptive (“Adaptability entails experimenting with various prompt formulations, phrasings, and temperature settings in order to establish a balance between creativity and concentration.”)
- Reflective (“Adopting a reflective perspective enables users to evaluate the performance of their AI model based on user feedback and their own assessments, identifying areas for improvement and adjusting their approach accordingly.”)

Effective prompt engineering directly influences the quality of evaluations made by LLM-as-a-judge. For instance, a well-designed prompt clearly specifying evaluation criteria, context,

and expected output format can significantly enhance the accuracy and consistency of the LLM's assessments.

2.5 Related Work

Existing Approaches

One of the existing approaches is the use of personal trainers, these are usually qualified professionals who can offer services such as creating a training plan tailored to your needs, coaching you through the movements, and creating diets to follow. Some of the pros of using a personal trainer include, personalized training and diet plans which can be discussed with another human and changed accordingly, in-person coaching through new or dangerous techniques in the gym, motivation as they can be in the gym with you encouraging you to work hard, and the reputation behind the qualifications. However, they can often be very expensive, as shown in the problem definition section averaging between £50 - £150 a session (thefitnessgrp.co.uk, 2023), despite the qualifications they may have it could still be very difficult to find a PT (Personal trainer) with knowledge about the specific sport (especially if it's a lesser known sport in the UK). Furthermore, PT's have normal working hours which may not fit around your schedule if you have to work unusual hours or multiple jobs. Finally, some people may not be comfortable interacting with a person they don't know when it comes to their health and fitness, especially those who come under the neurodivergent category.

Another existing approach is the use of the internet, specifically websites such as YouTube. YouTube is a video sharing platform that can be uploaded to by anyone with an account and viewed by anyone (account only needed to watch if flagged as an 18+ video), this means it contains a lot of information and because sports are so popular worldwide it attracts both experts and amateurs. Some of the positives of YouTube are the ease of access for most videos can be watched by anyone at any time from a computer, console, or smartphone, this can be especially useful if one wanted to review a video while at the gym or training ground. It can also contain information on lesser-known sports as it's a global platform and therefore there is much more demand. Finally, the production of these videos can be easier to absorb and entertaining to watch, involving the watcher more in the community. However, one of the cons to YouTube is the lack of credentials and proof, as I said before anyone can make a channel and upload a video, this means there is often no real way of knowing if the advice/training plan is effective. Another issue is that the amount of content on YouTube can make it hard to find the right video. The way creators make money is by users viewing ads

before their video, this means many creators can use clickbait titles or reupload old content, this makes it very hard to find good advice for a popular topic, such as workouts for football.

Another approach is the use of AI coaches, for example FitnessAI (FitnessAI, 2024) or ChatGPT. FitnessAI is a mobile app that generates personalized workout plans using artificial intelligence. It tailors' routines based on inputs, such as fitness goals, experience levels, equipment availability, and individual progress. It focuses on strength training and bodybuilding. FitnessAI has a great design and very easy to use interface, this makes it brilliant for people who have little to no knowledge and just need a quick and easy way to create a workout plan for them. However, it does cost £87.99 a year, while this is not a huge amount of money compared to personal trainers or even a gym membership it may still stop many people from using it. Furthermore, the app doesn't let you focus on a specific sport, and some reviews note how the workouts don't seem to be much better than what you could find online for free.

3 Methodology

3.1 Introduction

In this section I cover the tools, technologies, planning methods, and evaluation techniques I use throughout my project. I hope by the end you will understand what I used and why I chose them, as well as why they were good choices for this project.

3.2 Project Planning

Overview

Instead of using a formal development methodology like Waterfall, I chose a more flexible iterative approach that suited my work style and the changing requirements of my project better. I started by creating a high-level checklist of my main goals and objectives, then broke each section down into smaller, manageable tasks. This made it easier to focus on what needed to be done and helped me estimate timeframes more accurately. The list itself evolved as the project progressed, and I updated it regularly based on feedback from my tutor and any new challenges that came up, part of the list can be seen below.

Research has shown that iterative approaches are effective as they allow for better scheduling and control over activities (Sommerville, 2011), this is especially useful in my project as I am working alone instead of with a team where it can make coordination and progress estimations more difficult. Using this approach also allows me to get more feedback than I would with a traditional waterfall approach, this is particularly beneficial for a student project as the marking criteria is not as well defined so having direct feedback from one of my markers allows me to stay on track and put my time into the correct work.

FYP Plan	<input checked="" type="checkbox"/> Mathematical optimization
Synopsis	<input checked="" type="checkbox"/> GSP
Problem Definition	<input checked="" type="checkbox"/> WSGP
<input checked="" type="checkbox"/> Finalize problem	<input checked="" type="checkbox"/> Table
<input checked="" type="checkbox"/> Find main causes	<input checked="" type="checkbox"/> Conclusion
<input checked="" type="checkbox"/> Impact?	<input checked="" type="checkbox"/> References
<input checked="" type="checkbox"/> Why is it a problem?	Supporting Code
<input checked="" type="checkbox"/> References	<input checked="" type="checkbox"/> Fitness
Aims and Objectives	<input checked="" type="checkbox"/> LLM-as-a-judge
<input checked="" type="checkbox"/> Aim	<input checked="" type="checkbox"/> Prompt engineering
<input checked="" type="checkbox"/> Objectives	<input checked="" type="checkbox"/> References
Background Sources	Related Work
<input checked="" type="checkbox"/> Proof of problem	<input checked="" type="checkbox"/> Existing approaches
<input checked="" type="checkbox"/> Existing approaches	<input checked="" type="checkbox"/> References
<input checked="" type="checkbox"/> Methods	Design
<input checked="" type="checkbox"/> Datasets	High-Level
<input checked="" type="checkbox"/> References	<input checked="" type="checkbox"/> Flow chart
Approach	<input checked="" type="checkbox"/> Sequence diagram
<input checked="" type="checkbox"/> Overall methodology	Algorithms
<input checked="" type="checkbox"/> Plan	<input checked="" type="checkbox"/> GA flow chart
<input checked="" type="checkbox"/> Ethics	<input checked="" type="checkbox"/> GA pseudocode
<input checked="" type="checkbox"/> References	<input checked="" type="checkbox"/> SA flow chart
Evaluation	<input checked="" type="checkbox"/> SA pseudocode
<input checked="" type="checkbox"/> Software testing	<input checked="" type="checkbox"/> AGO flow chart
<input checked="" type="checkbox"/> Application testing	<input checked="" type="checkbox"/> AGO pseudocode
Background	Supporting Code
Overview	<input checked="" type="checkbox"/> Fitness checklist
<input checked="" type="checkbox"/> Introduction	<input checked="" type="checkbox"/> Fitness flow chart
<input checked="" type="checkbox"/> Goal	<input checked="" type="checkbox"/> LLM prompts
Problem	<input checked="" type="checkbox"/> Starting workout pseudocode
<input checked="" type="checkbox"/> Proof of problem	<input checked="" type="checkbox"/> Random person pseudocode
<input checked="" type="checkbox"/> Abstraction	GUI
<input checked="" type="checkbox"/> References	<input checked="" type="checkbox"/> Wireframe
Solutions	<input checked="" type="checkbox"/> Colours/fonts
	Database
	<input checked="" type="checkbox"/> People schema

Figure 4 – A section of my list used for planning

In the example of the main list you can see the breakdown of a couple of my tasks, you may notice the use of a couple of different colours, I used orange if the section is in progress or not quite finished, as you can see despite being past the design phase it is orange as I haven't created a data-flow diagram. Generally, if I “completed” a section and didn't need a part of it I would delete it from the checklist however, because I want to use a data flow diagram in the design section of the write-up, I plan to do it later and therefore kept it in.

Below you can see a timeline I created with the high-level dates/times I planned for. Naturally I did not follow this perfectly as certain parts took longer or shorter than expected. For example, my implementation went into my testing time due to new features I wanted to add, and the evaluation phase was therefore shorter than expected. I also want to point out the reason for background being given so much time, this was because it was over the winter break, this meant due to family commitments I didn't have as much time to put towards it on a day-to-day basis, it also covered the November/early December period where I had lots of

coursework for my other modules due so my FYP took a back seat for that time.

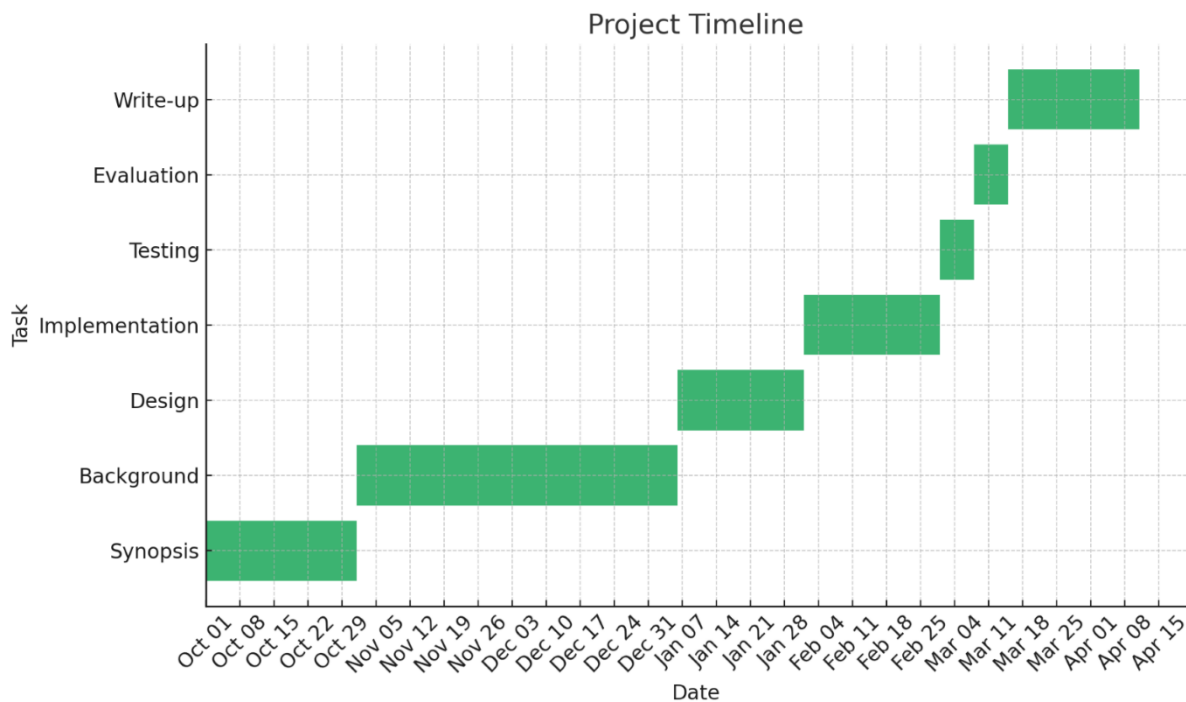


Figure 5 – Showing my project timeline

Design

For my design I decided to use mainly flow charts, this is because due to using Python class diagrams did not work particularly well and as the three algorithms (genetic algorithm, simulated annealing, and ant colony optimization) I chose to implement where very commonly used there was plenty of recourses available. Another reason I wanted to use flow charts is because they leave the implementation open and flexible as I knew I would be adjusting the algorithms to work with my specific use case.

For my database design I chose to use basic schemas as they let me design the database quickly and easily so that I could then work on finding the data. They also allow me to be flexible in case I wish to add or remove columns, assuming I did this before implementing.

Finally, for the GUI I didn't spend much time on it as it was mainly to be used in demonstrations and as a proof that this could work as a fully developed application. Therefore, I used wireframes to decide on the layout, and then following the do's and don'ts of designing for accessibility from the UK government website (GOV.UK, 2016) I chose simple colours which are easy to read in different levels of light and provide enough contrast, and a font which again could be read easily.

Implementation

My implementation was possibly the largest single part of my project, this meant it needed to be well thought out and planned. I once again used my checklist approach to create the tasks I would need to complete and then worked in cycles of one week centred around my meetings with my tutor on Thursday. This allowed me to continually add to the work I had done and ensured I didn't miss any opportunities. For most of my implementation I used a list on the note's app on my phone, I unfortunately worked by deleting the specific task once it had been completed so I don't have a full list of detailed tasks however this is an example taken from a screenshot I took.

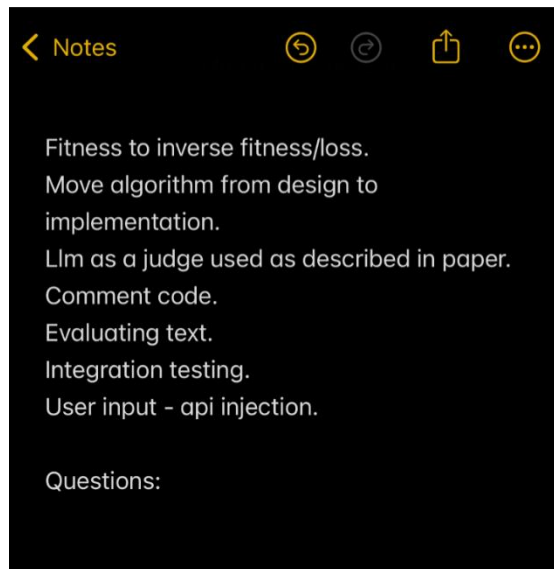


Figure 6 – A screenshot of a list from my phone

This helped me keep up with any smaller tasks or improvements I had been given in my meetings or had thought of whilst not having access to my laptop. The iterative nature of my planning allowed me to add features I did not originally plan on having such as the random person function which made it significantly easier to test my work without having to enter information every time. I also chose to use it with my final testing as it allowed me to test on a loop, but this will be covered in another section.

Testing and Evaluation

For my testing I decided on using unit testing, integration testing, and profiling for the software testing. To plan the unit testing I created a table with the tests I wanted to run on each function as well as my expected outcome as to have a pass/fail criteria. For integration testing I also used a table with the expected flow to follow and how the program should perform, this was done for each algorithm and used the GUI for inputs as this is how it would work in a full application. For the profiling this was run on each of the three algorithms as to

find any bottlenecks with the view of improving the efficiency of these functions, again the results are saved in a table format.

To test the outcome of the three algorithms I decided to use an automated testing sequence which would run each of them on the same random persons data capturing the time taken, final fitness, plan, user information, and results from using LLM-as-a-judge. I chose to this over using human experts as I can run many more tests and have the outcomes evaluated quickly and automatically on a loop. The data I am receiving from the LLM is a rating on how well the difficulty of the workout reflects the user's skill, a rating on how sport focused the workout is, and an overall rating (All on a scale of 1 to 5 where 5 is perfect and 1 is not good), and finally some additional comments. All this data will be saved to a table so it can be sorted and evaluated easily.

3.3 Technologies Used

Design

Technology	Reason
Draw.io - For creating my flowcharts and other diagrams.	Simple, easy to use, can be run online, good presets, no licences needed.

Table 2 - Technologies used for design

Implementation

Technology	Reason
Visual Studio Code – For my coding environment.	Used lots previously so set up to my preferences, easy to use, good support for all languages.
Python 3.13 - For most my code.	Easy to use, lots of useful libraries (e.g. Tkinter), good for rapid development and testing, lots of AI integration support.
SQLite – For my database.	Lightweight, easy to integrate with Python, no need for external servers. Great for small projects.
Tkinter/customTkinter - For my GUI.	Built into Python, so no extra installs. CustomTkinter made it look more modern with minimal effort.
OpenAI – For all LLM integration.	Easy to integrate with Python, reliable, and cost-effective for small usage.

Table 3 - Technologies used for implementation

Testing

Technology	Reason
Pytest – For my unit tests.	Lightweight and easy to use for unit tests. Well-documented and integrates nicely with Python.
CProfile – For profiling.	Built into Python and helped me profile the performance of algorithms.

Table 4 - Technologies used for testing

Supporting

Technology	Reason
Microsoft word – For all checklists, write-up, and some general planning.	Familiar, easy to organize documents, easy backup to cloud for access on multiple machines.
Microsoft Excel – To create database's before importing, and timeline.	Familiar, easy to organize documents, easy backup to cloud for access on multiple machines.
PowerBI – To create graphs and explore the data from testing.	Familiar as used in my placement, almost any chart/graph available, good customization.

Table 5 - Technologies used for supporting activities

3.4 Ethics

During my project I have not used any real person data, nor have involved real people in any way. All my code is written by myself unless otherwise specified within the comments or dissertation, I used ChatGPT for help with understanding errors, data generation (As described in my design section), and some code generation (For example the code used to generate random people for my people table in the database) but it is all referenced fully within this document and my code using comments. My write up was also written by myself unless otherwise specified, I used ChatGPT throughout for help with structure and brainstorming but never directly used or copied any output unless referenced. The ethics pro-forma is attached within appendix B.1, and I have read the Brunel guidelines on research misconduct, plagiarism, and AI use.

4 Design

4.1 Introduction

In this section I will show how I designed my program to meet my project aim. I'm trying to achieve a program that will take in information from the user such as sport, age, and skill and will, using the most effective algorithm, output a balanced sport-specific workout plan. My design covers the three main parts of my code, the database, the algorithms, and the GUI as well as more detailed sections on specific parts of the back end which were more complicated and important, these are the fitness function, and the LLM integration (specifically the prompt design). As mentioned in my methodology section I mainly use flow charts and pseudocode throughout this section, this is due to their flexibility and how well they work with my chosen language of python, I also use database schema and wireframes for the database and GUI sections respectively.

Workout Plan Output

The output of my program will be a full workout plan based on the user's inputs. This will be formatted as described in the table below, I have chosen not to specify what days the user should work out on as to keep it flexible around their schedule, I have instead taken this into account within the fitness function and made sure to design it to avoid working the same muscle two days in a row. This should mean the user can do any amount of days in a row and still have the time to recover (Assuming they are eating well and getting sufficient sleep, however these are out of the scope for my program), I can include links to other resources which cover rest, diet, and more workout information. I have also chosen not to include sets and reps with each exercise, this again can be covered in the additional resources, I have chosen to do this because in recent studies it has been shown that as long as the user is training close to failure the reps and sets don't make a significant difference (Schoenfeld et al., 2021), this is especially true with people who are newer to resistance training but still true for "well-trained" people (Schoenfeld et al., 2014). The real reason to choose different rep ranges is based on how easy the user can get to failure, for example it may be easier to get to failure doing 3 sets of 20 repetitions at a lower weight than 2 sets of 5 repetitions of a higher repetition, but usually depends on the users confidence and the specific exercise.

Leg Day	Back Day	Core Day
<ul style="list-style-type: none">- Side Lunge- Quad Stretch- Barbell Squats	<ul style="list-style-type: none">- Cat-Cow Stretch- Supine Twist- Deadlifts	<ul style="list-style-type: none">- Plank- Bridge- Crunch

- Leg Press	- Lat Pull-down	- Leg Raises
- Calf Raises	- Barbell Rows	- Russian Twist
- Hamstring Curls	- Shrugs	- Side Plank

Table 6 - Example workout plan

System architecture

The user will directly interact with the GUI, this is where they will enter any information required and where the workout will be displayed. Once they have entered their information and clicked start the information will go through validation, this will make sure it is all realistic and nothing is left blank, it will also help with any security concerns such as an injection attack. Once this has been passed the users data will be saved to a database, and the program will get the sport information from a different table in the database based on what the user has chosen. Now the algorithm will be run (the specific algorithm will be chosen after a comparison of the three chosen to test), this will output the workout plan back to the GUI which will display it to the user. If the user decides the plan isn't right, they can rerun the program and add any comments in the comment box, this can include any injuries or personal preferences and will send the comments to GPT-4o which will then follow a prompt instructing it to return the comments in a usable format for my fitness function, this will then rerun the algorithm and return a new workout to the user.

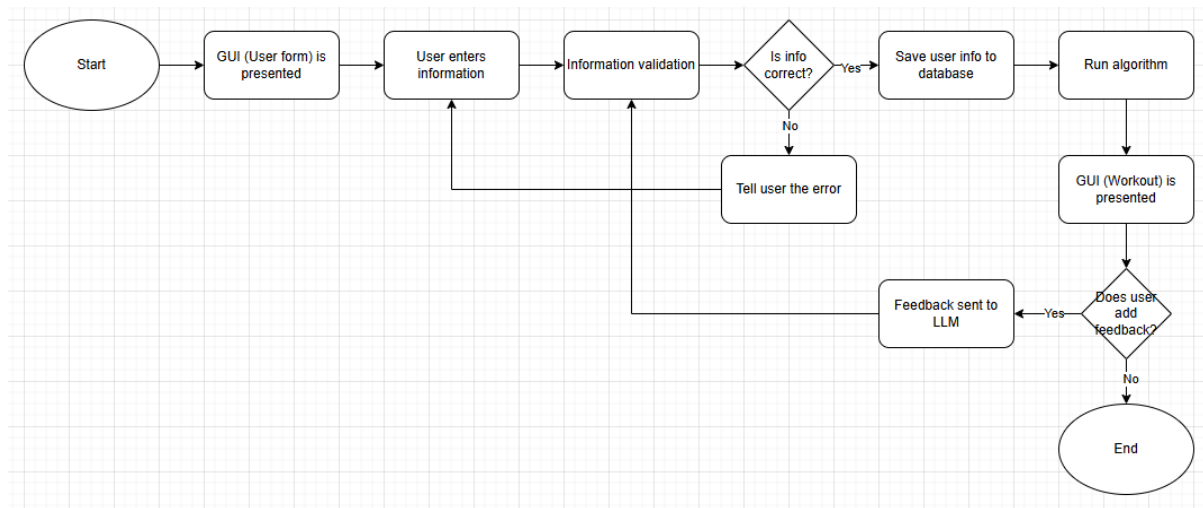


Figure 7 – Flowchart showing the flow of my program

4.2 Database Design

Introduction

In this section I will describe and explain my database usage, this includes why I chose SQLite, how I designed the tables, examples of the data stored in the tables, and how this data will be used in the program.

I chose SQLite mainly for its easy set-up, simple local use, and ease of use. As it doesn't require a database server like MySQL (which was the database I was going to use initially) I am able to set it up within VSCode and deal with all data manipulation (insertion, deletion, etc) within the same file. As my program is going to be stored and run fully locally from my laptop I don't have to worry about authentication or access control. I am also not going to have a particularly large dataset of sports, exercises or people, this means I shouldn't face performance issues using it where another solution would work better. Finally, as it uses SQL which I am familiar with from second year the use of it should be easy, there is also plenty of good documentation and help online as its the second most popular language used by professional developers according to a study from Stack Overflow (Stack Overflow, 2024).

As the tables will not directly interact with each other and will be used to store data that will be called within the code I have chosen not to use an entity-relationship diagram and instead will describe and explain each table and its contents one by one. How they connect should become clear in the algorithm sections.

Sports Table

This table will hold information about all the sports the user can pick. I decided the most important information for choosing the correct exercises would be the amount each sport is anaerobic/aerobic and the main muscle groups used. This is shown by the table below.

Column Names	Type	Example
id	Integer, PK	14
Sport Name	String	Rugby
Anaerobic	Float	0.75
Aerobic	Float	0.25
Muscle Groups	String	Quads, Shoulders, Core

Table 7 - Sports table design

The ID will be the primary key for this table, it will be a simple integer. The sport name will be a string and will be used to find the sport from the database, this will be chosen by the user. Anaerobic and aerobic will both be floats that together will add to 1, this means they can be seen as a percentage of how much each sport leans to either side, I chose this as I believe most sports are a little anaerobic and aerobic but having random numbers may make future calculations within the fitness function more difficult to make as they will add to different totals. Finally, the muscle groups will be a string of the main muscle groups and muscles used within the sport, they will be separated by commas to make them easy to separate in the code if need be.

Exercise's Table

This table will hold all the exercises I will use to make up the workout plans and accompanying information which can be used in the fitness function to fit them in. The information saved will be its difficulty, the equipment needed, type (anaerobic/aerobic), and the muscle groups it works.

Column Names	Type	Example
id	Integer, PK	98
Exercise Name	String	Zercher Squats
Difficulty	integer	3
Equipment	String	Barbell
Type	String	Anaerobic
Muscle Groups	String	Quads, Core

Table 8 - Exercise table design

Once again, the ID will be an integer and the primary key. The exercise name will be a string and is what will be shown to the user once the database has been created. The difficulty is going to be an integer between 1 and 5, with 5 being very difficult and 1 being easy, this can be used with the user's skill to work out if the exercise is suitable for them. Equipment is a string and will be any equipment needed, if there is more than one entry it will be separated by a comma. Type is a string and will either be stretch, anaerobic, or aerobic, this means it will match the types from the sport or be shown as a stretch, I have kept this separate as it will mean when creating the number of exercises within each day the program will always include stretches. Finally, muscle groups are the same as the sports database being a string and separated by commas if there is more than one.

People Table

This is a table filled with information about users, it will primarily be used to test the algorithm and will be filled with made-up information, making sure to have the extremes. The information from the user will be their name, age, skill level, the number of sessions they can do per week, the length of these sessions, and the chosen sport.

Column Names	Type	Example
Name	String, PK	Caspar Ashworth
Age	integer	23
Skill	integer	4
Session Number	integer	4
Session Length	integer	60

Sport	String	Wrestling
-------	--------	-----------

Table 9 - People table design

For this table I will be using the name as the primary key, this is so it will replace their workout if they create a new one, therefore stopping the table being filled with redundant data. The age will be considered in a similar way to skill, as people who are older may struggle to complete some exercises. Skill is the main way I will keep the workout doable and not overwhelming for a new user, it is on a scale of 1 to 5 as to match the difficulty in the exercise table, with 1 being low/no experience, and 5 being lots of experience in the gym. Session number is how many sessions the user will be doing each week, and session length is how many minutes the user plans for each session. Finally, sport will correspond with the sport chosen and be the main variable that most of the fitness will work on.

Testing Table

This table will hold the testing results from my planned automated testing using LLM-as-a-judge. It will hold the algorithm name, the time taken to complete, the best fitness, the difficulty relevance score, the sport focus score, the overall score, LLM comments, the plan, and the user information.

Column Names	Type	Example
id	integer, PK	4
Algorithm	String	GA
Completion time	float	60.45
Fitness	float	0.015
Difficulty	integer	4
Focus	integer	3
Overall rating	integer	4
Comments	String	Good range of exercises, could use more shoulder work for the sport.
Plan	String	Shoulders Day: Triangle Pose with Twist, Shoulder Blade Squeeze, Medicine Ball Slams, Lunges, Leg Raises, Jump Squats, Bench Press, Pike Push-Ups

		Hip flexors Day: Camel Pose, Standing Calf Stretch, Battle Ropes Slams, High Knees, Sled Push, Swimming, Pike Push-Ups, Treadmill Incline Walk
User	String	Caspar, 23, 4, 4, 60 Wrestling

Table 10 - Testing table design

For this table I will automatically create a new id for each entry, this will act as the primary key. The algorithm name will allow me to filter and compare by algorithm in the evaluation section. Completion time and fitness are both useful for me to see how effective the algorithm is. Difficulty, focus, and overall rating are all on a scale of 1 to 5 and given via the LLM, this will be one of the main ways I score the workouts in mass as it would be very labour intensive to score each personally or with a human expert. Comments are also from the LLM and I will use in the evaluation for extra clarity. Finally plan and user are there in case I wish to look at the specific plans created in more detail.

How I will fill the tables

To fill the tables I will be using ChatGPT. I chose to do this instead of taking data from tables that already exist because the data available was often filled with redundant data (for my needs) and I would still have to manually add lots of my own columns, such as type (as this is created for my program specifically).

As I will be judging my program mainly on which algorithm performs better, I believe that making sure the data they are using is the same is more important than it being perfectly accurate. However, I will be checking the data that ChatGPT creates to make sure it has understood my prompt, and the data is accurate. I will be doing this by taking a random sample of each table and comparing the data to information online. For example, I will take a random row from the exercise table and use the Jefit (Jefit, 2024) database to ensure the data is correct. I will do this to 10% of each table.

I will then take the table generated and enter it into excel to go over any formatting details or errors, finally I will import it directly from excel into the SQLite database tables.

4.3 Algorithm Design

Number of Stretches/Exercises per day

As you will see in the future sections I will be generally creating a random initial population, as I wish for the workouts to include both stretching and exercises I will be specifying how many of each the workout will have each day, this will be done in a function before the initial population is made, it will also ensure the stretches come before the exercises in the workout plan.

To decide on the number of stretches and exercises I will take the session length, as chosen by the user, and take 20% of it for stretching and 80% for the other exercises. Then assigning 5 minutes to each stretch (This is calculated from doing 3, 1 minute hold of each stretch with some time to rest and move to next exercise) and 10 minutes to each exercise (Calculated by using an average time of each set being less than a minute on average, then allowing for at least a minute rest time in between, this means they could fit around 5 sets in the slot if needed although this is a high number of sets) I can find a good base for the amount of stretches and exercises the user could fit in each day. Of course, this won't be able to always be perfect, but I believe it matches the average and allows for extra time on each so most users should be able to complete all the full day in under the time they have limited themselves to.

Genetic Algorithm

The genetic algorithm will follow the normal structure as seen in the flow chart below. This was taken from the paper (ResearchGate, 2020a)

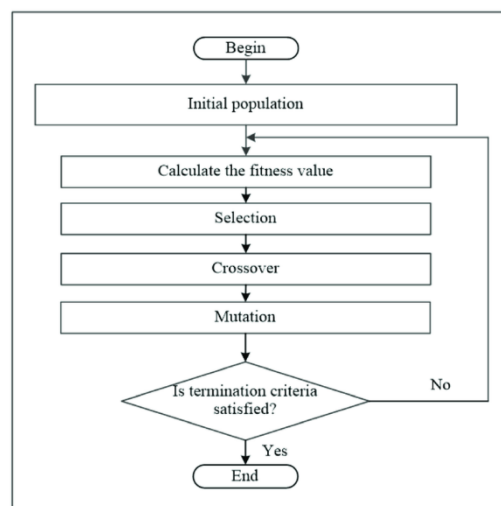


Figure 8 – Flowchart for genetic algorithms

For my use the individuals will be full workout plans (The length decided by the users inputs), the population will therefore be a defined number of workout plans. The initial population will

be workout plans filled randomly within their assigned stretches/exercises as describes previously.

Each individual in the population will then be assigned a fitness from my fitness function, this is described in detail later in the design section.

Next it will go through selection, I will be using tournament selection as this helps guide the population towards better fitness without always picking the best and therefore allowing for exploration and avoiding getting stuck in local optima. For the tournament size I will use 3, this will allow it to be computationally efficient for a medium sized population.

will then put the population through the crossover, here I will be using uniform crossover on each day of each plan. Uniform crossover mixes each exercise with a probability of 50%, this allows for maximum mixing between parents, also due to how the stretches and exercises are placed it means they stay in the correct place every time. I will be creating two children for two parents meaning the population size will stay constant throughout the process.

Finally, I will perform mutation on the population. This will again loop through every single exercise or every day of every plan and depending on the mutation rate chosen it will randomly switch an exercise with another from the database, again this will have to keep stretches to stretches and exercises to exercises, hence why I will go through each individually. This helps again with the exploration of the algorithm.

At the end of the process, I will save the best plan and its fitness, based on the fitness. This will be replaced if another plan in future populations is found and will be the final output of the algorithm.

For the genetic algorithm I will have to decide on the size of the population, the mutation rate, and the number of generations. For this I will create a function that will run the algorithm many times with different random parameters within a realistic range (size: 50-200, mutation rate: 0.01-0.3, generations: 100-500) and save the result with the best fitness, these parameters will then be used by the algorithm.

Simulated Annealing

My simulated annealing algorithm will be standard and follows the flow chart from (ResearchGate, 2011) shown below.

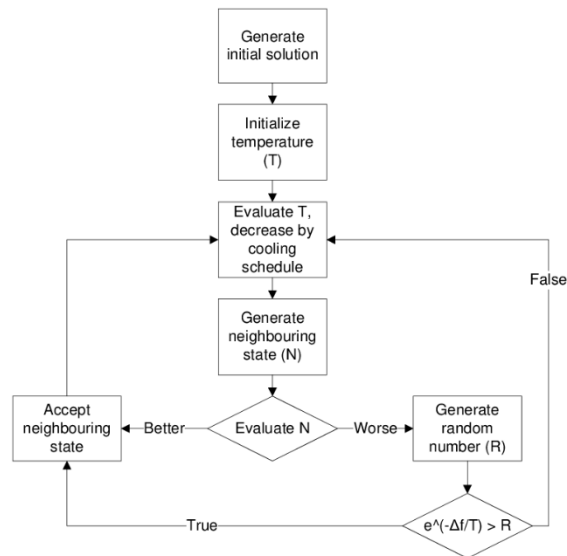


Figure 9 – Flowchart for simulated annealing

I will be reusing the initial population function from the genetic algorithm, however due to it creating a population of many plans where simulated annealing only requires one I will set the size to 1.

I will then initialize the temperature and the other parameters, these will be chosen in the same way as the parameters for the genetic algorithm, the ranges are initial temperature: 1000-5000, cooling rate: 0.85-0.99, iterations: 500-3000, and mutation rate: 0.1-0.3.

I will then lower the temperature by the cooling rate, this therefore means the algorithm is less likely to choose a plan with a worse fitness as it goes on.

I will then generate a neighbouring plan, this will be done using the mutation function from the genetic algorithm, hence why mutation rate is included in the parameters.

Using the fitness function I will evaluate the plan. If the fitness is better than it will save the new plan and repeat the algorithm. If it's worse than it will generate a random number, this will be compared with the exponential of the delta (which is the difference in fitness between the plans) divided by the temperature. This means it has a chance to be chosen even if the fitness is worse, especially if it's near the start of the algorithm as the temperature will be higher. This allows it to escape local optima and explore more.

This algorithm will repeat for the number of iterations chosen, returning the plan with the best fitness.

Ant Colony Optimization

Once again, my ant colony optimization will follow the flowchart below from (ResearchGate, 2020b).

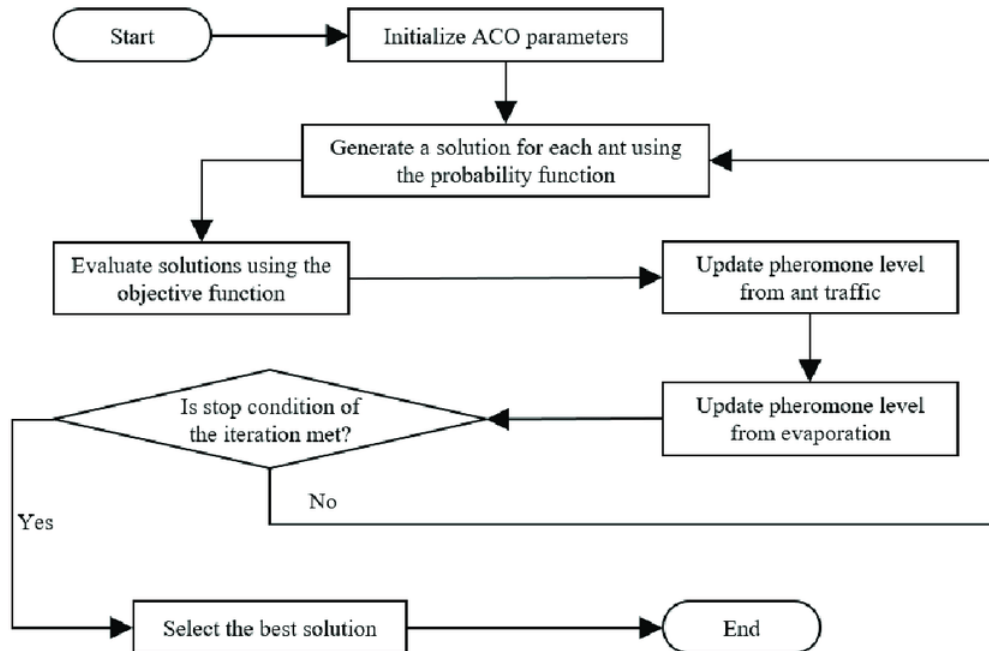


Figure 10 – Flowchart for ant colony optimization

First, I will set the parameters, once again these will be chosen through the same method as the genetic algorithm and the simulated annealing. The parameters and their ranges are ant number: 20-100, iterations: 10-100, evaporation rate: 0.05-0.3. In ant colony optimization there is also the pheromones and heuristics, pheromones for all exercises are all set to 1.0 at the beginning and depending on the evaporation rate and how often the exercise is visited will be updated, the heuristics are similar to a simplified fitness function applied to each exercise.

My heuristics will consider the difficulty of the exercise compared to the skill of the user, the muscle groups of the exercise compared to the sport, and the type (anaerobic/aerobic) compared to the sport. These do not change after they are set at the beginning.

Next each ant will create a plan based on a probability of choosing each exercise, decided by the pheromones and the heuristics.

I will then use the fitness function for each plan to evaluate it.

Next, the pheromones will be updated using a deposit factor (if exercise used this will be applied) and the evaporation rate if the exercise has not been used.

I will save the solution with the best fitness and return it as the output.

4.4 Fitness Design

Introduction

The fitness function is arguably the most important function in my program, it is used in all three algorithms to both select the best answer and directly in the population choice sections. It is generally the part of the system that will encourage the algorithms to follow my plan for a “good” workout plan. It takes in the plan and the user preferences and rates each day on how well the exercises match, then consolidating it and finding an average rating for the plan as a whole. I will be using penalty/reward-based inverted fitness, it will penalize the plan when it goes against constraints by increasing the fitness and reward it when it matches my objectives by decreasing the fitness. This means I can better fine tune it with many different objectives/constraints. Being an inverse fitness means lower is better, I will start at 1.0 and use multiplication to raise or lower it, this adds another level of tuning to it as not only can I change the number its multiplied by but also where in the function it takes place, as the sooner it happens the larger the number it will be effecting and therefore the bigger the difference it will make. Some of the time the fitness will be dynamic as it will be based of another variable as well, for example the user feedback with the sensitivity.

Objectives

It's very important I have clear objectives which I can incorporate into my fitness function. The table below shows each of the objectives, how I will use them, and how important they are on a scale of 1-10, this will help me decide on the initial number and order, although this may change during implementation and testing.

Objective	Description	Design	Importance
Exercise type match.	The exercises should generally match the sport type (aerobic/anaerobic).	I will work out the difference between the types for the sport as they are done as percentages and then penalize/reward the fitness based on the difference.	7
Muscle group match.	The exercise should match the muscle groups used in the sport.	Depending how many muscle groups match the	8

		fitness will be lowered or raised.	
User skill against exercise difficulty.	The difficulty of the exercise should not exceed the skill of the user.	The difficulty level will be compared to the skill level and fitness will be penalized or rewarded.	7
Duplicate exercises.	There should be as few duplicate exercises as possible within the plan.	I will track all the exercises and depending on the number of duplicates I will penalize the fitness.	8
Same muscle per day.	Generally, I want each day to focus on a particular set of muscles, this stops overworking in consecutive days and allows rest.	Saves muscle groups within a day and rewards the more of the same there are.	7
Sequential muscle overlap.	We don't want the same muscle group to be worked multiple days in a row as to allow for rest.	Save the main muscle groups of each day and compare, if they overlap then penalize fitness.	9
User feedback.	This is taken from the LLM based off the users feedback, it will be returned as a list of preferences/constraints with sensitivity on a scale of 1-5 based on how important.	Check through list and if muscle or exercise appears use sensitivity to raise or lower fitness.	9

Table 11 - Fitness function objectives

4.5 LLM Integration Design

Introduction

I've used LLM integration twice in my project, the first is for taking user feedback and returning it in a way my fitness function can use it to adjust the workouts, and the second for my testing to evaluate the workouts that the three algorithms return. I will be using gpt-4o for the user feedback and o3-mini for the evaluation. I am going to use an API call for both and have chosen

these models from OpenAI because they are quick and easy to set up, I don't have to have anything installed locally, fairly cheap for my purposes, and I can easily control my account (The money, how many calls etc).

As explained in my background and methodology in order to use the LLMs effectively the prompt is incredibly important, therefore this section will describe how and why I've deigned the prompts the way I have.

Model Choice

I've will two API calls to OpenAI in my project, one to process the user feedback and return it in a form my fitness function can then use, and another for LLM-as-a-judge which I covered in my background and will also cover further in my testing section. This section will cover what models I used and why.

For my user feedback function I used gpt-4o, this is because it's of course one of the newest and more powerful models provided and has very good contextual accuracy. This is important as the user feedback could come in any form and the task is to change their text into specific muscle groups, exercises, and sensitives. The cost (\$2.50 per 1M tokens) here was not as important to check as this won't be called as often in my testing, it is still relatively low cost.

For the testing API call I used GPT 3.5 turbo (o3-mini) as its slightly lower cost (\$1.10 per 1M tokens) yet still very capable of grading the tasks given a well thought out and accurate prompt. As this will run many more times during testing the cost was more important, and due to the task being repetitive and structured it should be sufficient for the task.

User Feedback Prompt

For the user feedback the input will be a sentence or two straight form the user up to 100 characters as this is enough for multiple constraints, preferences, or injuries and will save on the cost. It will return a list of any preferred exercises, preferred muscles, constraint exercises, and constraint muscles with a sensitivity on a scale of 1-5 where 1 is not very important and 5 is incredibly important.

Below is a table showing the structure of my prompt design, using context, instruction, input data, and output format with a description of what this should include in the right column.

Component	Description
-----------	-------------

Context	<p>It knows it's in a workout setting, looking at user feedback on a workout.</p> <p>It should be matching the feedback to muscles and exercises with sensitivity.</p>
Instruction	<p>It should check for any mention of injury as to give it a high sensitivity.</p> <p>It should map any body parts to muscles from the list provided.</p> <p>It should understand the sensitivity rating.</p> <p>It should strictly keep to muscles and exercises included in the lists provided.</p>
Input data	I give it the user feedback, the list of exercises from my database, and the list of muscle groups from my database.
Output format	<p>The output format should follow the exact layout I will provide, this matches the dictionary I will create to store the output.</p> <p>It must be in strict JSON format.</p>

Table 12 – User feedback prompt design

I will follow the CLEAR framework by (Springer, 2023) to craft my prompt, this means I will keep it concise throughout, structure it in a logical way as shown above, keep the output explicit as also shown above, possibly adapt it in testing, and reflective by testing throughout implementation. This also covers some of OpenAI's recommendations (OpenAI, 2024) on being clear, including context, and having a structured output.

Evaluation Prompt

For the evaluation prompt the input will be the workout plan, the users details, and the sport details. It will return scored on difficulty, sport focus, overall rating, and some additional comments. The scores will be on a scale of 1 to 5 where 1 is very bad and 5 is very good.

Below is another table showing the components of the prompt and the description as to what they should include.

Component	Description
Context	It knows it's in a sport/workout setting, analysing sport-specific gym workouts.

Instruction	<p>It should rate it on difficulty compared to the skill of the user, aiming for it to be well balanced, explain the scale.</p> <p>It should rate it on the sport focus nature of the workout, explaining the workout should be sport-specific but still balanced.</p> <p>It should give it an overall rating, taking into account the previous scores.</p> <p>It should add at least one sentence to describe the scores, and any additional improvements or praises.</p>
Input data	I will give it the workout plan, the user info, and the sport info from my database.
Output format	The output should follow an example I will include which matches the dictionary I create to hold the data. It should be in JSON format.

Table 13 - Evaluation prompt design

Once again, I will continue to use the CLEAR framework and OpenAI's suggestions to create the prompt and will test continuously during development. The values this prompt returns will be saved to the testing table in my database.

4.6 GUI Design

Introduction

For my GUI I will be using Tkinter within python. I chose this because its built-in, simple to learn, and quick to implement, as my GUI is mainly for presentation purposes to show what a full user experience could be like if my program was implemented properly my priority is speed and ease-of-use. Tkinter allows me to create the front-end fully within my python code without setting anything up other than importing it onto my python install, it also has lots of good documentation on the Github repository and other websites online making it easy to learn and use.

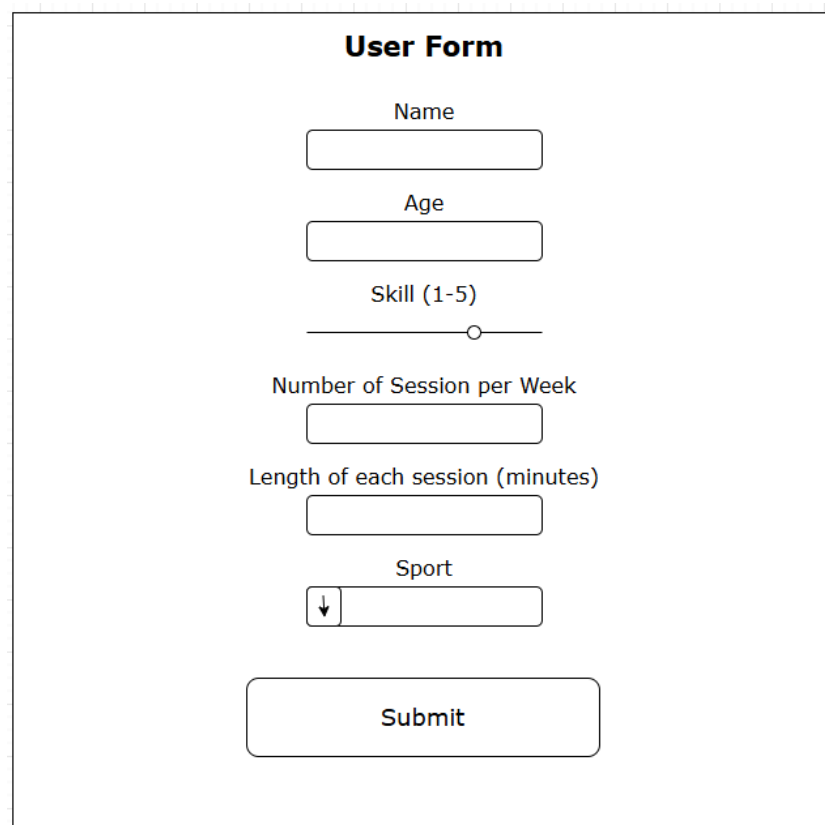
My GUI will consist of two parts, the user form and the workout plan output. The user form will capture information and save it to the people database described above, therefore it will ask for the users name, age, skill, session number, session length, sport, and any feedback/comments. The workout plan output just needs to show the best plan (according to the fitness) my program was able to create in an easy to read and understand way.

User Form

This is where I will collect the data from the user and where the user can choose to start the program, it should be clean, simple, and easy to use as well as following the UK governments guide for designing for accessibility (Government Digital Service, 2016) and implementing as many ways to limit user error as possible.

The form will need to include text boxes for the name, age, session number, session length. For the sport I will use a drop-down menu with all the sports from my database as to stop the user typing a sport I have not included yet, and for skill I will use a slider as to make it clear and keep them to using 1 to 5. For the other fields I will use my validation class to check for errors such as not being filled in, using words instead of numbers, and unrealistic entries (e.g. session length at 1000 minutes).

Below is the wireframe for my design, as you can see I kept it very simple as to make it as easy as possible for anyone to use.



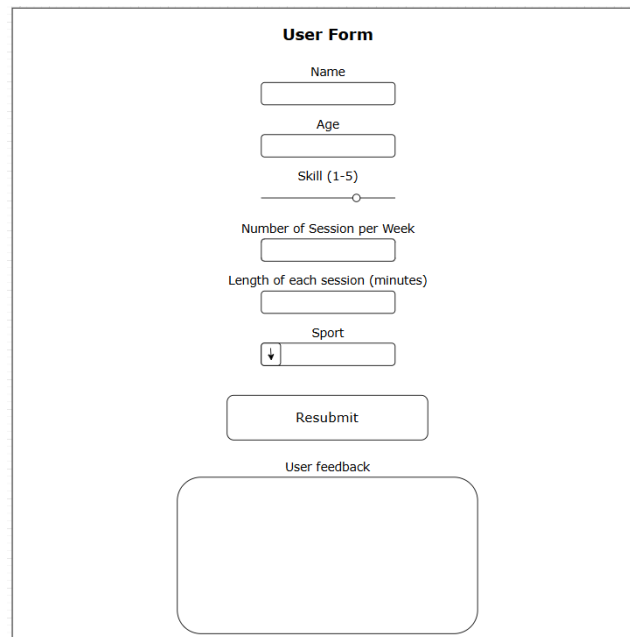
The wireframe shows a 'User Form' with the following elements:

- Name:** A text input field.
- Age:** A text input field.
- Skill (1-5):** A horizontal slider control.
- Number of Session per Week:** A text input field.
- Length of each session (minutes):** A text input field.
- Sport:** A dropdown menu with a downward arrow icon.
- Submit:** A large rectangular button.

Figure 11 - Wireframe of my GUI

For my font I will use Verdana, this is because is it clear and easy to read due to being designed specifically for screens (Microsoft, 1996), it also keeps a clean and professional look whilst being friendly and inviting. I will use a “dark mode” for my colours as it can reduce eye strain, focus user attention, and again gives it a clean, professional look.

Once the form has been submitted it will add the user feedback box to the bottom of the form as seen in the wireframe below. This allows the user to resubmit the form with feedback.



The wireframe shows a 'User Form' with the following elements:

- Name**: A text input field.
- Age**: A text input field.
- Skill (1-5)**: A horizontal slider control.
- Number of Session per Week**: A text input field.
- Length of each session (minutes)**: A text input field.
- Sport**: A dropdown menu with a downward arrow icon.
- Resubmit**: A rectangular button.
- User feedback**: A large, rounded rectangular text area.

Figure 12 – Wireframe of my GUI with user feedback

5 Implementation

5.1 Introduction

This chapter is split into two main parts, the first will cover how my code works, including the flow of each algorithm and explanations on each function. The second part will cover the story of my implementation, I will mention some of the bigger issues I faced and what I did to solve them. I hope that together these will help you understand my code and the core functionality, as well as why I chose to do it the way I have.

5.2 Functions

In this section I will go over each algorithm I've implemented, and the functions used, I will explain what inputs it takes, how it processes them, and then the outputs. I hope after this my code should be clear.

Below is a table with a quick description of each function for easier digestion. The algorithm column is where each function can be found in terms, if left blank the function is not called while in the algorithms function and instead called separately.

GA = genetic algorithm, ACO = ant colony optimization, SA = simulated annealing.

File	Algorithm	Function	Description
Main		randomPerson	Fetches a random user from the database.
Main		sessionSplit	Calculates the number of stretches and exercises each day, fills with placeholder.
Main		geneticAlgorithm	Main GA function.
Main		antColonyOptimization	Main ACO function.
Main		simulatedAnnealing	Main SA function.
Main		llmAsAJudge	Sends workout plan to ChatGPT for scoring.
Main		inputValidation	Validates user inputs and returns clean data or error.

Main		fillWorkout	Fills the plan with random exercises and stretches from the database.
Main	GA, ACO, SA	llmConstraints	Send user feedback to ChatGPT to generate constraints.
Main	GA, SA	startingPop	Generates a starting population of workout plans, uses fillWorkout.
Main	GA, ACO, SA	fitnessCalc	Calculates the fitness of each workout.
Main	GA	selection	Uses tournament selection to select from population.
Main	GA	crossover	Uses uniform crossover to create children out of parents.
Main	GA, SA	mutation	Randomly changes stretches/exercises based on mutation rate.
Main	GA, ACO, SA	restDays	Renames workout plans days based on muscle group.
Main	ACO	heuristics	Calculates score for each exercise.
Main	ACO	selectExercises	Selects exercises for plan based on heuristics and pheromones.
Main	ACO	updatePheromones	Updates pheromones based on plan fitness and evaporation rate.
GUI		submitForm	Collects user data from the form, validates it and starts algorithm.
GUI		runAlgorithm	Runs the algorithm in a new thread.
GUI		updateGUI	Updates GUI with workout and feedback box.
Database		insertpeopleData	Adds user data to the people table.
Database		viewPeopleData	Gets all data from the people table.
Database		getSports	Gets all data from sports table.
Database		getExercises	Gets all data from exercise table.

Database		getMuscleGroups	Gets all muscle groups from exercise table.
Database		getSportSpecific	Gets data about single sport.
Database		getExerciseSpecific	Gets data about single exercise.
Database		getExerciseType	Gets all exercise of a specific type.
Database		createTestingtable	Creates the testing table.
Database		insertTestingData	Adds testing data to testing table.
Database		getTestingData	Gets all data from testing table.

Table 14 - Functions

5.3 Main functions

This section explains how each of the algorithms works and each function in more detail. I have separated the fitness as its more complicated and important than the other supporting functions. If you wish to I encourage you to look at my code directly as well as reading this section, I have commented all of the functions in details and in appendix d is a table with information on each python file I have uploaded.

Fitness function

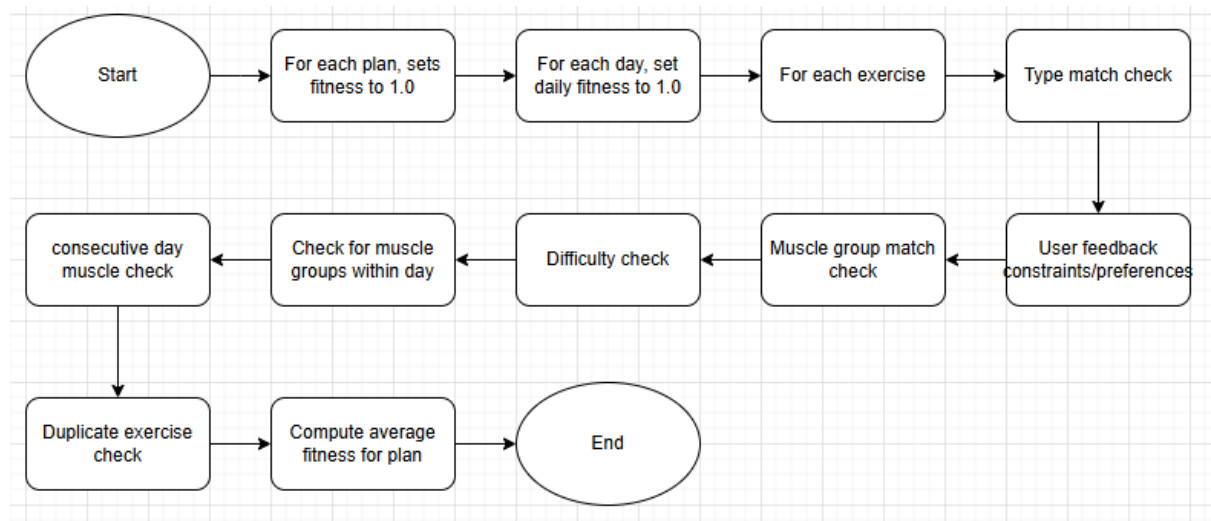


Figure 13 – Flowchart of the fitness function

As this function is used in all three of my algorithms and is arguably the most important part of each, I will use this section to describe the design of it in more detail. Importantly this is technically an inverse fitness, where the lower the better.

	Variable Name	Type
--	---------------	------

Inputs	sport, population, totalNumber, skill, constraints	String, Dictionary, Integer, Integer, dictionary
Outputs	fitness, muscleGroupDay	Float, Dictionary

Table 15 - Fitness function

I loop through each plan, setting the fitness to 1.0, each day of each plan, setting a daily fitness to 1.0 and each exercise.

First check is whether the exercise matches the type of the sport. As type can be stretch, anaerobic, or aerobic I first have to check for stretches and skip them as they are not relevant here. I then calculate the difference between anaerobic and aerobic from the sport data, if the exercise type matches the sport type, then the daily fitness will be lowered by the difference between sport types. Doing it this way ensures that the exercises chosen will match the balance of anaerobic and aerobic from the sport, this is important as a balance of both is generally required for most sports.

Next, I take the constraints and preferences from the user feedback, this has been parsed through ChatGPT to simplify and make it usable. If there are any exercises or muscle groups present it will raise or lower the fitness based on the sensitivity, for example an injured arm will be turned into bicep, triceps, shoulder muscle groups and a sensitivity of 5, this will then raise the daily fitness by 1.5 $((\text{sensitivity} / 10) + 1)$ if the exercise hits these muscles. The same goes for preferences although the calculation is $\text{sensitivity} / 10$.

Now it checks the muscle groups for the exercise and compares against the sport. As each exercise and sport hits multiple muscle groups the fitness will be lowered depending on how many matches, if only one it will be lowered by 5% but if three match then it will be lowered by 20%.

Next is the difficulty check, this is not dynamic and just lowers the fitness by 10% if the difficulty is the same as or lower than the user's skill.

We now leave the exercises loop, so everything that follows checks for the full day.

After difficulty I check for the amount of each muscle group is present within the day, using the number of the highest count I lower the fitness using this calculation, $(1 - (\text{highestCount} / 10))$, this encourages the plan to train a single muscle group as much as possible within one day.

Now to combat the problem of having every day train the same muscle group I have implemented a check for consecutive days. Here I compare the list of muscle groups from the previous day to the current day, if there are no overlaps I lower the fitness by 30%, if there are under 2 then I lower by 10%, otherwise I don't lower it at all.

Now we leave the daily loop so everything forward is for the full plan.

Final check is for duplicate exercises within the plan, here I count the number of duplicates and use this calculation, $\text{penalty} = (\text{duplicates} / \text{totalNumber}) / 10$ to decide the fitness penalty. This is applied to the daily fitness.

Finally, I find the average daily fitness and use that as the plan fitness which is returned.

Genetic Algorithm Function

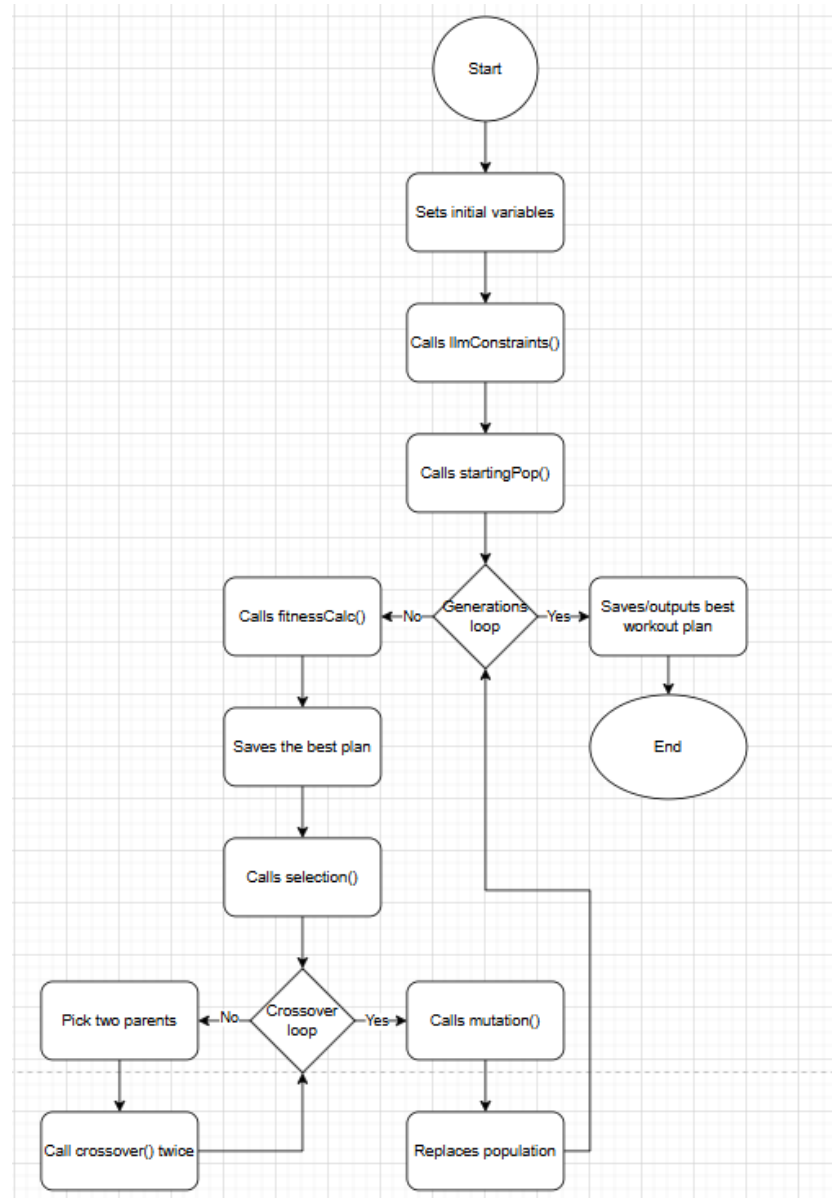


Figure 14 – Flowchart of my genetic algorithm

First, I set the size, mutation rate, and generations. Having it at the top and separated makes it easy to find and edit during testing.

I then call `llmConstraints()`.

	Variable Name	Type
Inputs	feedback	String
Outputs	constraints	dictionary

Table 16 - `llmConstraints` function

This is a function which is where the users feedback is sent. Within the function the constraints dictionary is created, the prompt is written, and the API call to OpenAI will take place. It then returns the constraints dictionary which will be used in the fitness function.

Next, I call startingPop().

	Variable Name	Type
Inputs	size, workoutPlan, stretchNumber, exerciseNumber	Integer, dictionary, integer, integer
Outputs	population	dictionary

Table 17 - startingPop function

This is the function that creates the starting population. It does this by taking the size of the population, workout plan (which is from sessionSplit() and is a workout plan dictionary of the correct size filled with stretch/exercise as a filler), stretchNumber (which is the number of stretches in each day), and exerciseNumber (which is the number of exercises in each day). Within the function it loops through each plan and calls fillWorkout() function which in turn loops through each day and fills with a random stretch or exercises from the database. It then returns the starting population.

Now the start of a loop which will repeat based on the number of generations set at the beginning. In the flow chart “no” indicates it has not looped through each generation yet, and “yes” means it has finished.

I call fitnessCalc().

	Variable Name	Type
Inputs	sport, population, totalNumber, skill, constraints	String, dictionary, integer, String, dictionary
Outputs	Fitness, muscleGroupDay	dictionary, dictionary

Table 18 - fitnessCalc function

This is the function that will find the fitness of each workout plan in the population. It does this based on the sport, muscle groups, skill level, any constraints/preferences, and general rules

such as not overworking a particular muscle group two days in a row. As this is one of the most complicated functions and is used in all three algorithms it is explained in full in its own section. It will return the fitness for each plan in a dictionary, and the main muscle group worked on each day of each plan for naming reasons.

Next, the algorithm will save the workout plan with the lowest fitness under a new variable.

I then call selection()

	Variable Name	Type
Inputs	population, fitness, tournamentSize	Dictionary, Dictionary, integer
Outputs	selectedPopulation	Dictionary

Table 19 - selection function

This is the function that will select the best plans of the current generation based on their fitness using tournament selection. It loops through each workout plan, taking three at random and using tournament selection to find the plan with the lowest fitness, this is then chosen and added to the new population, the loop keeps the new population the same size and the current population. This new population is what's returned and used through the rest of the algorithm.

Next is the crossover loop. This is the start of a loop which will go through each workout plan 2 at a time, this is to perform the crossover function at a rate in which the size of the population stays the same. Same as before “no” means the loop will continue as it has not iterated through all the population, and “yes” means it has and will continue to the next section.

It now will pick two random workout plans and rename them as parent1 and parent2, these will be used in the crossover function next.

It then calls crossover() twice.

	Variable Name	Type
Inputs	Parent1, parent2	Dictionary, dictionary
Outputs	child	dictionary

Table 20 - crossover function

This is a function that performs the crossover between two parents creating a single child. It does this by taking the two “parents” from the previous step and looping through each exercise

within each day, every time it will pick randomly (50/50) which parent to take the exercise from and add to the child. It does this for all the exercises (this includes stretches, due to it being done one by one the stretches will always stay as stretches and exercises do the same). It then returns the child. This is done twice in a row so that the number of plans in the population stays the same.

Call `mutation()` function.

	Variable Name	Type
Inputs	nextPopulation, mutationRate	Dictionary, float
Outputs	nextPopulation	dictionary

Table 21 - mutation function

This is a function that will perform mutation on the population. This is done by looping through each exercise in each day or each population, every time it will randomly (Based on the `mutationRate` float) switch the exercise with another random exercise from the database. Again, to keep the ratio of stretches to exercises the same it will only switch a stretch with another stretch and exercises with another exercise. It then returns the population with all the mutations included.

Just before the end of the loop the dictionary `nextPopulation` is saved over the dictionary `population` so that the loop can start again and repeat.

Finally, once the loop has completed for the amount of generations specified the plan saved as `bestPlan` (based on the lowest fitness) will be returned from the function.

Ant colony optimization function

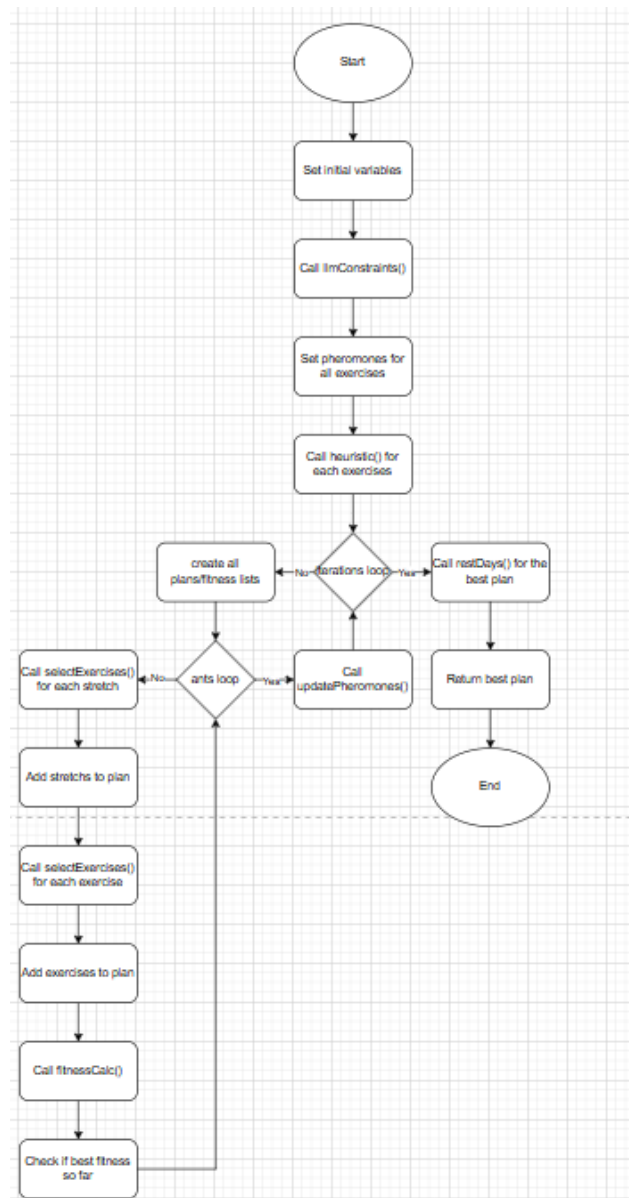


Figure 15 – Flowchart of my ant colony optimization algorithm

First, it once again set the numAnts, iterations, and evaporationRate. For the same reason as the GA, it makes it easy to edit these commonly changed variables at the start.

It then calls llmConstraints().

It then loops through every exercise from the database and set the starting pheromone. This will be updated through the iterations based on fitness.

Call heuristic() for all exercises.

	Variable Name	Type
--	---------------	------

Inputs	exercise, muscles, sportType, skill, difference	String, String, String, String, float
Outputs	heuristic	float

Table 22 - heuristic function

This function calculates the heuristic for each exercise based on the skill/difficulty, muscle groups, and the type (anaerobic/aerobic). It does this by first setting the weighting for each of these so it can be edited to change which is most important, it then directly compares the skill of the user to the difficulty, calculates how many muscles in the exercise match the sport, and finally if the type matches the sport type. It returns the heuristic for the exercise, this is repeated for every exercise and saved in a list of heuristics.

Now we get to the start of the loop that will repeat for however many iterations where set at the start of the function. Much like the loops in the GA “no” represents there are more iterations to be done and “yes” indicates it has finished.

Next, it creates a list to hold all the plans created in the future and another to hold all the fitness’, this will be used to find the best plan.

This is the start of another loop, it represents a single ant per loop and so will repeat for the number saved under antsNum, each ant will create a plan so it's like the population number. The “no” and “yes” work the same as iterations loop.

Call selectExercises() for each stretch.

	Variable Name	Type
Inputs	pheromonesStretch, heuristics, visited	List, List, set
Outputs	idx	String

Table 23 - SelectExercises function (Stretches)

This is a function that depending on the heuristic and the pheromones will pick an exercise (for this example specifically a stretch). It does this by first checking if the exercise has been visited already, if it hasn't then based on a probability created from the heuristic and pheromone of the exercises (As well as weighting set at the beginning of the function). It will then pick a random

exercise out of all of them based on the probabilities. This exercise will be returned. This function will be called as many times as stretches needed for the plan. For each stretch returned from `selectExercises()` it will be added to the workout plan. This will be done for each day.

Call `selectExercises()` for each exercise.

	Variable Name	Type
Inputs	pheromonesExercise, heuristics, visited	List, List, set
Outputs	idx	String

Table 24 - SelectExercises function (Exercises)

This is the same as before however now using only exercises and not stretches.

Again to add exercises to the plan.. These are done separately so the number of each is always the same on each day as calculated from the `sessionSplit()` function. I then call the `fitnessCalc()` function. Here the function will check if the new plan has the lowest fitness so far, if it does it will be saved separately as `bestPlan`. The fitness of this plan will also be saved.

Call `updatePheromones()`.

	Variable Name	Type
Inputs	pheromonesStretch/Exercise, allPlans, allFitness, evaporationRate	List, List, List, float
Outputs	pheromonesStretch/Exercise	list

Table 25 - updatePheromones function

This is a function that updates all the pheromones based on evaporation rate and the fitness of their plan. It does this by first updating each from `evaporationRate` and creating the deposit rate, it then goes through each exercise in each plan and updates it again based on the deposit rate and the fitness/best fitness. This means the plans with the best fitness reflect that on each exercise, so they are more likely to be picked in future iterations. This is done separately for stretches and exercises again.

Call `restDays()` for the best plan.

	Variable Name	Type
--	---------------	------

Inputs	bestPlan, muscleGroupDay	Dictionary, dictionary
Outputs	bestPlan	dictionary

Table 26 - restDays function

This is a function that names the days based on the most prominent muscle group being exercised. It does this by taking in the best plan and the muscle groups for each day (This includes how many times each appear). It then renames each day to make it clearer for the user in the output and make it clearer for quick testing. It then returns the renamed plan.

Simulated annealing function

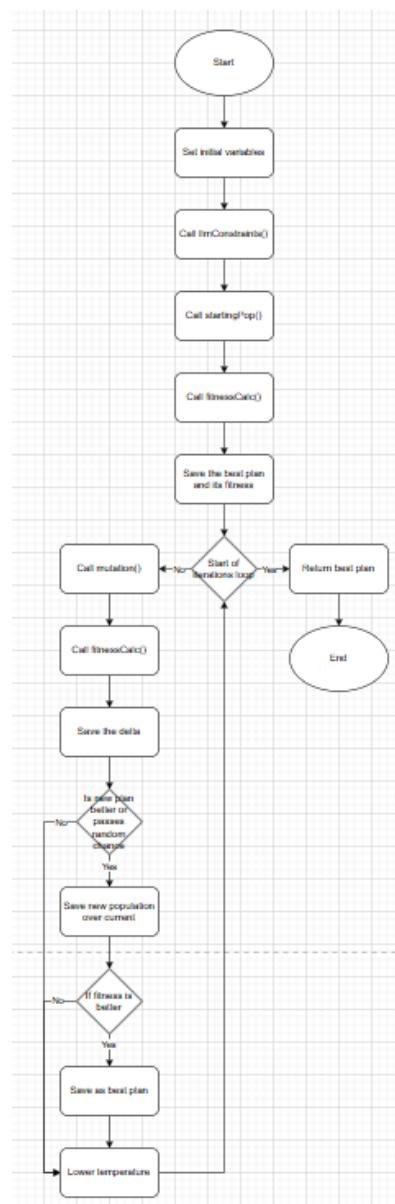


Figure 16 – Flowchart of my simulated annealing algorithm

First, I set the size, initial temperature, cooling rate, iterations and mutation rate. These are all variables that can be changed often to optimize the algorithm so having them easily accessible is useful. Size here is set to 1 meaning the population will only have 1 plan, this is because it works differently to the genetic algorithm however the startingPop() function is made to work with both. I then call llmConstraints(), startingPop() and fitnessCalc() again.

Next, I create the variables that will hold the best plan and its fitness, as we have just started it is filled by the initial population.

Start of iterations loop. This will loop over as many times as specified in the iterations variable. As before “no” indicates the loop has not repeated the specified amount if times and “yes” means it has finished (for the flowchart). I then call mutation() and fitnessCalc().

Next, I save the delta. The delta is the difference in fitness between the current plan and the newly mutated plan. It is used as part of the selection in the next step.

Now I have an if statement which will decide whether the new plan is saved and replaces the current plan or not. It is decided first on if the fitness is better (lower), or by random chance if a random generated number is smaller than the exponential of delta over the temperature, this means it has a random chance of taking it even if the fitness is worse depending on the temperature. This code can be seen in the picture below.

```
#If the new plan is better or the random number is lower than the exponential function (The worse the new fitness the less likely)
if delta > 0 or random.random() < np.exp(delta / initialTemp):

    #Save the new plan/fitness over the current plan/fitness
    currentPopulation = neighbourPopulation
    currentScore = neighbourScore

    #If the fitness is lower then save as best plan/fitness
    if currentScore < bestScore:
        bestPopulation = currentPopulation
        bestScore = currentScore
```

Figure 17 - Picture of simulated annealing code

If it passes the if statement it will save the mutated plan over the current plan and carry on using that.

Next there's another check to see if the fitness really is better than the current, as the previous if statement allowed for it to pass on chance. If the fitness really is better than it will save it separately as the best plan, in case we don't get back to a better fitness in future iterations.

Development of a Smart Sports-Specific Workout Generator Using Optimization Algorithms At the end of each iteration, I lower the temperature by the cooling rate, this means as it goes on its less likely to let a plan with a worse fitness be saved as the new plan.

Finally, after all the iterations the best plan and its fitness is returned.

5.4 GUI

To create my GUI I chose to use Tkinter and custom Tkinter as described in the design section it's easy to use, built into python, and have good documentation available.

Below is a flow chart showing how a user would interact with the GUI.

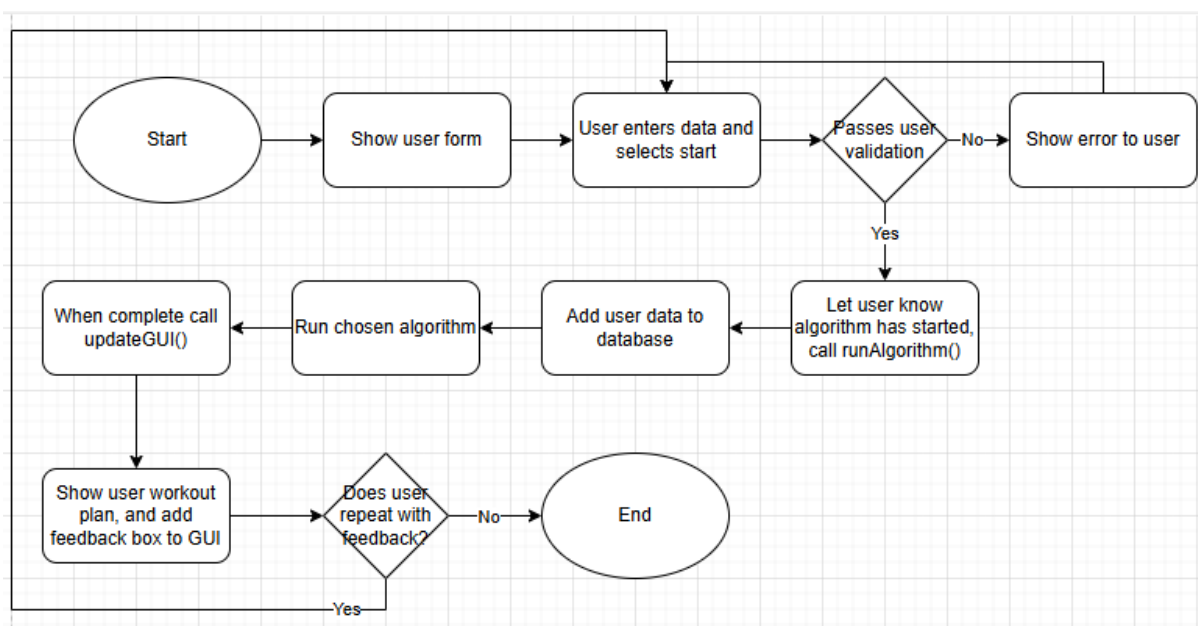


Figure 18 – Flowchart of my GUI flow

When the program starts the user form is presented to the user, this is seen in the image below.

Figure 19 – Screenshot of my GUI

Next, the user will enter their information and click submit, I am using a slider for the experience section and a drop-down box for the sport as to help lower chance of user error. The experience slider corresponds to the skill variable, I felt experience is easier to answer than skill for a user.

This will call the inputValidation function in main, here I check for everything shown in the table below, for the first error it finds it will return an error message that will be shown to the user, this can be seen below.

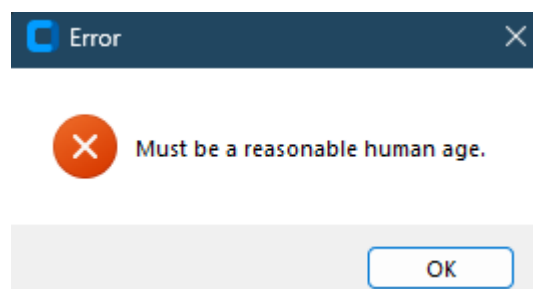


Figure 20 – Screenshot of an error message from my program

Variable	Check
name	If empty
age	If 0 or below
age	If above 100

age	If integer
skill	If 0 or below
skill	If above 5
sessionNumber	If 1 or below
sessionNumber	If above 14
sessionLength	If below 20
sessionLength	If above 180
sport	If empty
feedback	If longer than 100 characters

Table 27 - Input validation checks

If it passes user validation the submit button will change to say “processing...” and it will call the `runAlgorithm()` function on a different thread, this means you will still be able to move and interact with the user form box while the algorithm is running, although you won't be able to submit again as button will be greyed out.

In the `runAlgorithm()` function it will first add the user data to the people database then run `sessionSplit()` and the chosen algorithm, once its finished it will update the GUI.

In `updateGUI()` the workout will be shown in a new window as seen in the image below, and the user form will be updated with the user feedback box, this can also be seen below.

The user is now able to change their details and run the algorithm again, if chosen this new workout will show in another new window and won't replace the current workout.

User Form

What is your name?

How old are you?

Experience in the gym (1-5)

How many sessions can you do each week?

How long will these sessions be (in minutes)?

What sport are you training for?

Complete!

Workout Feedback box

Click the submit button again to regenerate workout with feedback!

Figure 21 – Screenshot of my GUI with feedback box

Workout			
Shoulders Day (Day 1)	Shoulders Day (Day 2)	Shoulders Day (Day 3)	Core Day (Day 4)
Shoulder Blade Squeeze	Downward Dog	Chest Stretch	Spinal Twist Stretch
Standing Shoulder Stretch	Eagle Arms Stretch	Triceps Stretch	Lat Stretch on Stability Ball
Mountain Climbers	Medicine Ball Slams	Battle Ropes Slams	Standing Calf Raises
Wall Walks	Dead Hang	Plank Hold	Jump Squats
Weighted Plank	Landmine Press	Stair Climbing	Rowing Sprint
Sledgehammer Slams	Pike Push-Ups	Sledgehammer Slams	Hanging Leg Raises
Wide Push-Ups	Face Pulls	Standing Calf Raises	Bicycle Crunches

Figure 22 - Screenshot of a workout in my GUI

5.5 Database

My database file may seem somewhat incomplete, it is set up to do anything required within the algorithms but as I set it up mid-way through the implementation some of the initial set up, for example creating and filling the people, exercise, and sport databases were done in different files. This will be covered in detail in the second part of this chapter, here I will cover the functions that are used within my code briefly.

As shown in the table at the start there are 11 different functions in this file, 8 of them are used to get data from tables in the database. I have one for each table which gets all the data from it and one which gets all the data from the muscle group column in the exercise table, then there are 3 which get specific information from tables. There is only one function that was used to create a table, this is the testing table which I created at the end of my development. Finally, two functions that insert data into tables, one for testing and one for people table (user data).

5.6 Story

Environment and data set up

First thing everyone has to do with a coding project is set up the environment, luckily I already had visual studio code set up with my preferred extensions and a virtual python environment set up from earlier testing with LLM integration. I set up my folder structure as using a different file for each algorithm, I thought this would be the best way of organising my code, although quickly discovered it was much easier to keep them all together in my “main.py” file due to the amount of shared functions.

At the start I explored lots of different sources for my sport and exercise data, many of which had lots of information which I would not be using in my program, because of this and the amount of formatting I would have to do I decided to instead make use of ChatGPT (For the first time in this project) to ask it to create the data. As the real test of my program is the comparison of the algorithms and how well the full system works together, I decided that the sport and exercise data did not have to be perfectly accurate as it would still be able to be used, as I described in a previous chapter I still checked the full list of both to make sure it was passable and no huge mistakes had been made.

To set up the SQLite database I made a new python file purely for creating the three tables (sports, exercises, and people) and for filling them with data. To fill them I copy and pasted the data from chatGPT into excel where I could format it as I wished. I then used pandas to read the tables from the excel file and copy the data directly into my SQLite database.

Genetic algorithm

Once the databases were filled I could start on my first algorithm, I decided to implement the genetic algorithm first as I thought it would be the most difficult and time-consuming (and I was correct). I mainly followed this guide (DataCamp, n.d.) of course writing and editing the code myself as the problem being solved is drastically different. This is where I ran into some of my first major difficulties as the fitness function at this point didn't include any check for duplicate exercises or encouraging the same muscle group on a single day. To fix this I added them to the list of objectives and solved them by adding the methods I described in the fitness function explanation of this chapter. This is also shown in the picture of my code below.

```
#Counts the number of each exercises
exerciseCount = Counter(allExercises)

#Goes through the count of each exercise and saves the amount of duplicates there are
duplicates = sum(count - 1 for count in exerciseCount.values() if count > 1)

#If there are any duplicates then the fitness is raised by an amount calculated on the number of duplicates (More duplicates = higher fitness)
if duplicates > 0:
    penalty = (duplicates / totalNumber) / 10
    dailyFitness = [value + penalty for value in dailyFitness]
```

Figure 23 - Picture of duplicate exercise fix from fitness function

This worked great but then very quickly I discovered the problem of hitting the same muscles every day, which would not allow for rest and recovery, causing me to add another objective and fix to the fitness (The fitness became a continuously evolving function with new objectives being discovered every couple of days).

GUI integration

Once I had a decently working genetic algorithm I decided to start on the GUI, as described in previous sections I decided to use Tkinter (At this point I wasn't using custom Tkinter). The initial set up didn't take too long but I ran into some problems with the sport choice, specifically at this point it was just a text box, of course this didn't work very well, not only because not all sports are in my database but also many of them are formatted in a way a user would not copy, for example sports like swimming had two versions swimming (Sprints) and swimming (Distance). To solve this issue, I decided to change it to a drop-down list and fill it with all the sports directly from my database. This is also when I created the database.py file to store the functions as up until now I had been just putting all the SQL code into the functions directly. So, I set up the new file and created some of the basic functions that I had been using already to simplify and organise my code better and more efficiently. I then used one of these functions to get all the sports from the database for my drop-down menu, this also means it will automatically update if I decide to add or remove any sports.

Another issue I had was the GUI would freeze when the algorithm was running, this of course looked terrible and meant the “processing...” label update I had put in to inform the user that is was working was not updating, when it froze the window couldn’t even be moved around the screen. To fix this I made a new function where the algorithm was called and any other back-end process (updating database etc) and ran it on a new thread. This worked perfectly and now the user can interact with the window and move it about to their hearts content.

I later started using custom Tkinter, this was when I went back to try and make it look “presentable” as well as easier to read and use. Custom Tkinter has a great documentation (customtkinter, n.d.) explaining how it works, this made it very easy to change what I needed to and design it as I chose.

LLM user feedback integration

Now the user feedback LLM integration, the process of setting it up was fairly simple thanks to OpenAI’s quick set-up guide (OpenAI, 2024), however creating a prompt which worked consistently wasn't so easy. The call would often return in the wrong format and using muscle groups or exercises which I was not using in my database, to fix the second issue I decided to upload a copy of all the exercises and muscle groups from my database into the prompt, I also made sure to specify how I wanted the output formatted with an example. Changing to a better model solved a lot of the inconsistency issues as well.

I also ran into issues implementing the feedback into my fitness function due to not separating the constraint and preferences into different rows instead having them on the same row with constraints first. This was fixed by of course adding more rows, and while doing that I decided to add a sensitivity row to have a more accurate dynamic fitness. Below you can see an example of this in my code.

```
#If there are items in the constraintsExercise column
if constraints['constraintExercise']:

    #Takes the data and saves under variables
    exerciseData = constraints['constraintExercise'].lower().split(',')
    sensitivity = int(constraints['constraintExerciseSens'])

    #Reduces the fitness based on if any exercises match and scaling on sensitivity
    if exercise in exerciseData:
        fitness[day] *= (sensitivity/10 + 1)
```

Figure 24 - Picture of constraint exercise part of fitness function

Finally, I added a functionality to skip the API call if the feedback was left blank, saving time and money.

Ant colony optimization and simulated annealing

At this point of my project, I had set up so many useful functions that both ant colony optimization and simulated annealing were made very easy to implement as they both used a lot of the same functions as my genetic algorithm. Of course, I had to edit some of the functions to work with single plans instead of a population, but much of this could be solved by filling a “population” with just one plan.

Testing

For testing I had to create some new functions and code in order to automatically test all three algorithms at once. I had already filled the people table with fake people (generated by ChatGPT) and had creating a function that picked a fake person at random, of course I had to manually edit some of the people as to make sure they covered a larger range of sets/reps for example. For my testing function I would use this to pick a random person, this same person would be used for all three algorithms in a row, timing how long each one took, then sending the results to my llm-as-a-judge function, created for this purpose and using much the same logic as my user feedback call but with a different model and slightly modified prompt.

This would work well although it would take a couple of minutes for each run, I then remembered my GUI and decided to run the three algorithms at the same time but on different threads to speed it up, now each loop only take as long as the longest algorithm.

Another part of my testing was finding the best parameters for each algorithm, for this I created a new function for each algorithm in a new python file, made for testing functions. These functions would run 50 times and choose random values within a range, saving the best after each run.

Finally, I set up my unit testing and profiling and ran them, although I will talk more about the results in the testing chapter.

6 Testing

6.1 Introduction

This chapter should showcase how my program performed and explain the results. I will be using unit testing, integration testing, profiling for my software testing.

6.2 Unit testing

Introduction

Unit testing is a software testing technique where I can simulate the use of individual functions/components in my code and validate they return the expected results. I will be using automated white-box unit testing as I know how the code works and am directly testing its flow.

I used Pytest for my unit testing as its built in, easy to use, and the documentation online in combination with how popular it is made it simple to troubleshoot. In this section I will show the tests I used, my results, and how I fixed some of the issues it found with my code.

Tests

Below is a table with every unit test in my code, and the result when I first ran it, as well as the results. As you will see the majority of the unit tests passed, one of the reasons for this is due to me implementing it at a similar time to user validation, this meant when I thought of the test, I was able to add it directly into my code before running the tests.

Name	Description	Result	Fix
testSessionSplit	Test that the output of this function splits the session into the expected number of stretches/exercises.	Pass	No fix
testNegativeSessionSplit	Enter a very small session length number to make sure it doesn't return anything below 0.	Pass	No fix
testStartingPopulation	Test to make sure the population is the correct length, and the days include the correct number of stretches/exercises.	Pass	No fix
testFillWorkout	Test to make sure the correct number of stretches/exercises are used, and they are all valid Strings.	Pass	No fix
testSelection	Test to make sure the size of population stays the same and the result is a plan that was entered.	Pass	No fix
testCrossover	Tests the child is the same length as the parent and each exercise can be found in one of the parents.	Pass	No fix

testMutationChange	Tests to make sure the output is different to the input is mutation happens.	Pass	No fix
testMutationNoChange	Test to make sure nothing happens when mutationRate is 0.	Pass	No fix
testRestDays	Tests whether the name assigned is correct.	Pass	No fix
testEmptyName	Tests error handling with an empty name.	Pass	No fix
testEmptyAge	Tests error handling with empty age.	Pass	No fix
testInvalidAgeNegative	Tests error handling with negative age.	Pass	No fix
testInvalidAgeTooHigh	Test error handling with age over 100.	Pass	No fix
testInvalidSkill	Test error handling with skill level not between 1 and 5.	Pass	No fix
testEmptySkill	Test error handling with empty skill level.	Pass	No fix
testInvalidSessionNumberLong	Test error handling with more than 14 sessions a week.	Pass	No fix
testInvalidSessionNumberShort	Test error handling with 0 session a week.	Pass	No fix
testEmptySessionNumber	Test error handling with no session number.	Pass	No fix
testInvalidSessionLengthShort	Test error handling with session length under 20 minutes.	Pass	No fix
testInvalidSessionLengthLong	Test error handling with session over 3 hours long.	Pass	No fix
testEmptySessionLength	Test error handling with empty session length.	Pass	No fix
testEmptySport	Test error handling with empty sport.	Pass	No fix
testValidInputs	Test with all valid inputs to ensure no error.	Pass	No fix
testFitnessCalc	Test fitness function runs with mocked database and doesn't crash and returns float value.	Fail, a couple of times	Fixed, described below table
testLlmConstraints	Test the results of some simple user feedback returns the expected muscle groups and exercises.	Pass	No fix
testGeneticAlgorithmSmoke	Runs a small plan to check the function returns a valid plan and fitness.	Pass	No fix
testAntColonyOptimizationSmoke	Runs a small plan to check the function returns a valid plan and fitness.	Pass	No fix
testSimulatedAnnealingSmoke	Runs a small plan to check the function returns a valid plan and fitness.	Pass	No fix

Table 28 - Unit test results

Results

The fitness function failed due to the mock databases not being filled fully, even though this was a mistake with the unit test it still made me add error handling to the fitness function in case database doesn't include muscle groups required.

The fitness functions since been edited in multiple places to check that any database calls return what's expected, and if not, it will use a placeholder value that won't affect the fitness itself so it's able to complete despite the issues.

I also ran a coverage test using pytest which returned the result of 81% of main.py code is covered in the tests I have used. I believe this covers most of the main functions, how they work as well as some error handling for the functions that use direct user input such as `inputValidation()`, I didn't add much error handling tests into the functions within the algorithms because these do not take any data that hasn't been through validation already.

6.3 Functional testing

Introduction

For this section I tested the full flow of the system to make sure it works as intended without bugs or crashes. I go through the GUI three times, each time I use the same information as shown below, and each time I will add the feedback after the first workout is returned. On each attempt I will use a different algorithm as the back end.

Name	Age	Skill	Session Number	Session Length	Sport	Feedback
Caspar	23	4	3	60	mma	"I don't like doing squats"

Table 29 - Functional testing input

Genetic Algorithm

The genetic algorithm took 247.73 seconds first and 206.98 seconds second, it produced the workouts below, the first is before feedback and the second is after.

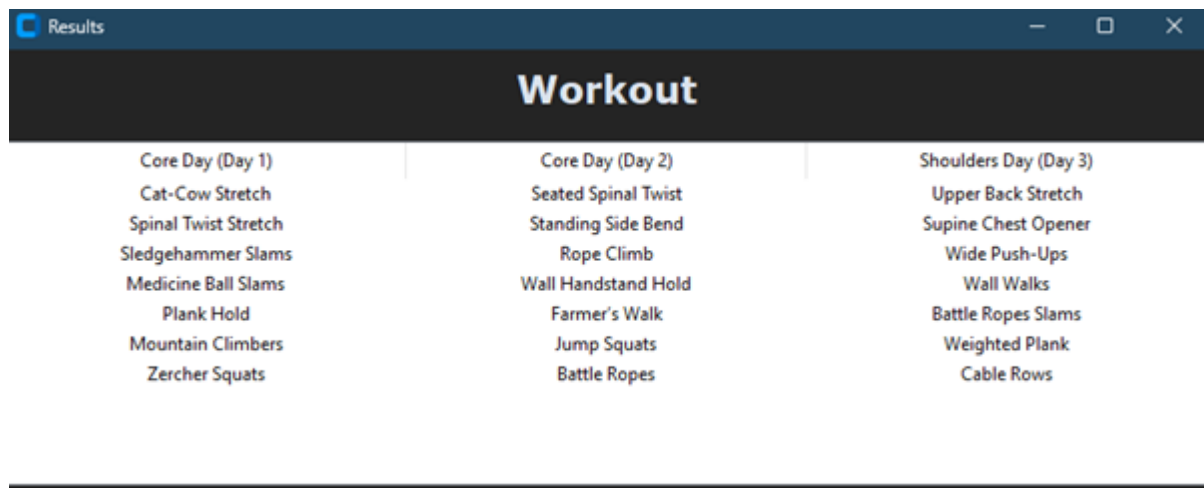


Figure 25 – Screenshot of a workout plan from my program before user feedback (GA)

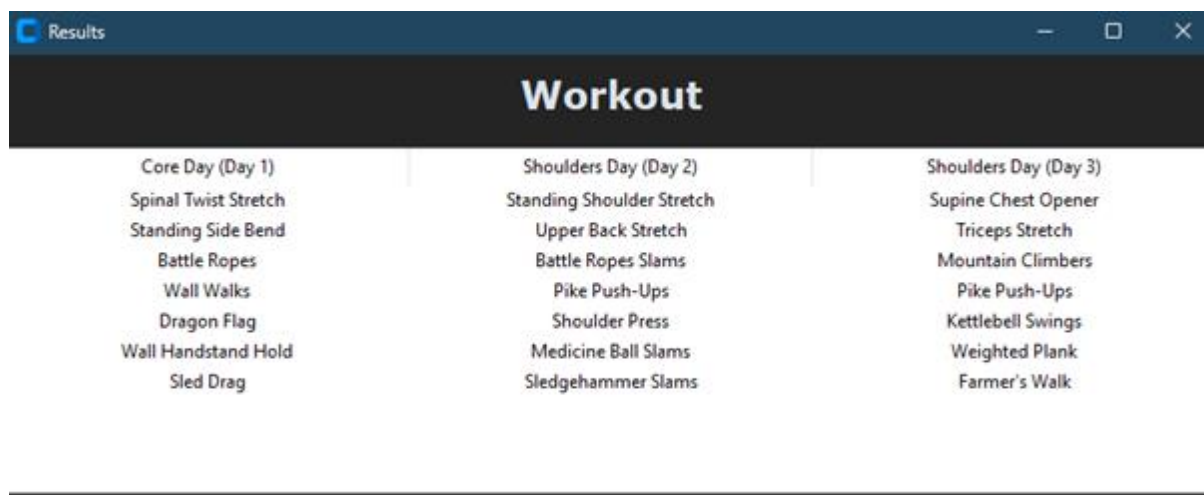


Figure 26 - Screenshot of a workout plan from my program after user feedback (GA)

From the two workouts we can see that not only was the system working without any crashes, but it has the correct amount of stretches/exercises in the correct order (stretches first) and the feedback was working as there are no squats in the second workout.

Simulated Annealing

Simulated annealing took 1.69 seconds first and 4.77 seconds second, it produced the workouts below, the first is before feedback and the second is after.

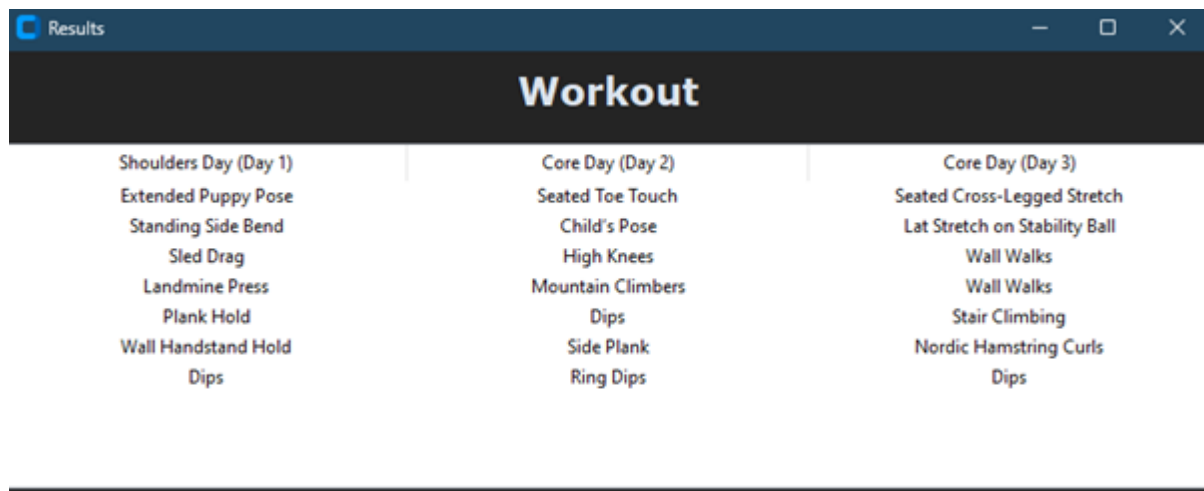


Figure 27 - Screenshot of a workout plan from my program before user feedback (SA)

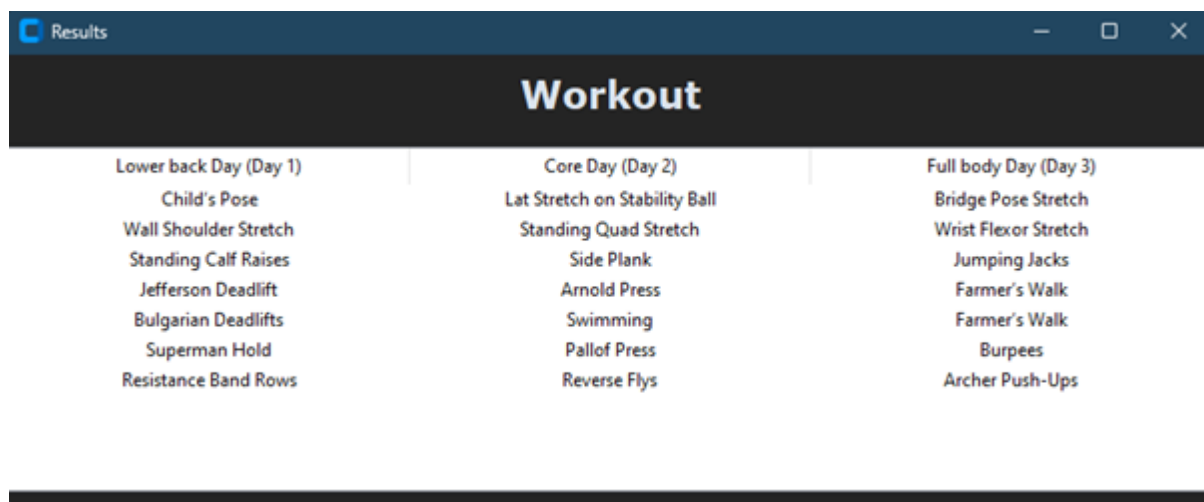


Figure 28 - Screenshot of a workout plan from my program after user feedback (SA)

This test once again shows the program working fully, of course the time taken was much lower and we can see the workouts both contain some double exercises, this is probably due to the parameters as changing these may have caused the time to increase but probably would have found a lower fitness without the doubles.

Ant Colony Optimisation

Ant colony optimisation took 15.36 seconds first and 29.44 seconds second, it produced the workouts below, the first is before feedback and the second is after.

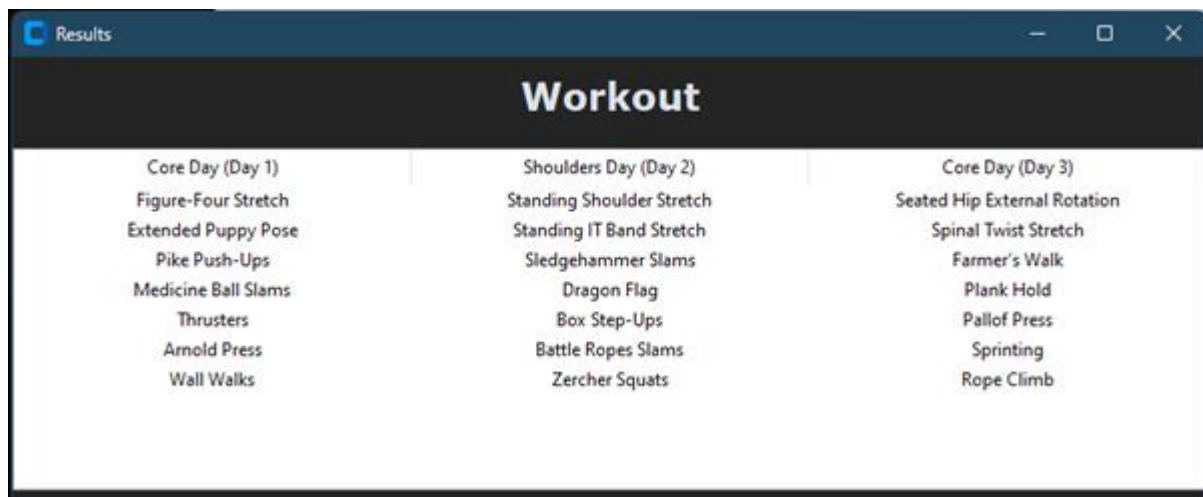


Figure 29 - Screenshot of a workout plan from my program before user feedback (ACO)

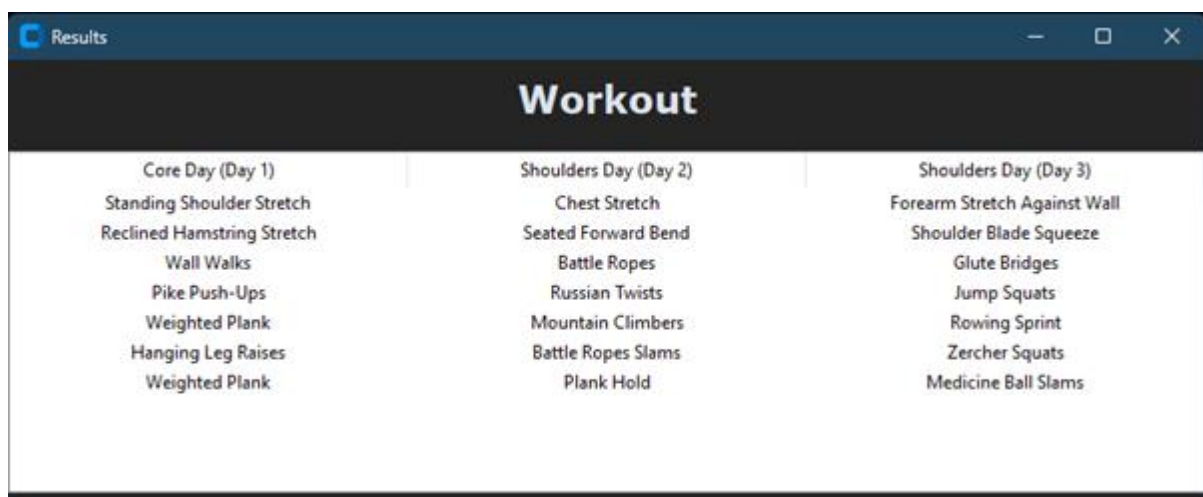


Figure 30 - Screenshot of a workout plan from my program after user feedback (ACO)

Once again the task was completed with no errors, however again due to being much quicker than the genetic algorithm it seemed to not quite find the best fitness in the second run as it still includes squats on day three. This could potentially be fixed by changing parameters.

6.4 Performance evaluation

Profiling (Without Feedback)

To test the computational efficiency of my algorithms I used profiling and timing. This means I can find any performance bottlenecks. I used cProfile to automate profiling on my code as its built into Python, running it on each of the three algorithms. This shows me how much time is being spent on each function overall and per call. I used pStats to view the results.

Each test was performed using the same inputs, these were:

GA:

```
workoutPlan = ('Day 1': ['stretch', 'stretch', 'stretch', 'exercise', 'exercise', 'exercise', 'exercise', 'exercise', 'exercise'],
                 'Day 2': ['stretch', 'stretch', 'stretch', 'exercise', 'exercise', 'exercise', 'exercise', 'exercise', 'exercise'],
                 'Day 3': ['stretch', 'stretch', 'stretch', 'exercise', 'exercise', 'exercise', 'exercise', 'exercise', 'exercise']),
stretchNumber = 3,
exerciseNumber = 6,
sport = "american football",
totalNumber = 9,
skill = 3,
feedback = "")
```

Figure 31 - Performance evaluation inputs for my GA

SA:

```
stretchNumber = 3,
exerciseNumber = 6,
sport = "american football",
totalNumber = 9,
skill = 3,
feedback = "",
workoutPlan = ('Day 1': ['stretch', 'stretch', 'stretch', 'exercise', 'exercise', 'exercise', 'exercise', 'exercise', 'exercise'],
                 'Day 2': ['stretch', 'stretch', 'stretch', 'exercise', 'exercise', 'exercise', 'exercise', 'exercise', 'exercise'],
                 'Day 3': ['stretch', 'stretch', 'stretch', 'exercise', 'exercise', 'exercise', 'exercise', 'exercise', 'exercise'])
```

Figure 32 - Performance evaluation inputs for my SA

ACO:

```
stretchNumber = 3,
exerciseNumber = 6,
sport = "american football",
totalNumber = 9,
skill = 3,
feedback = "",
sessionNumber = 3)
```

Figure 33 - Performance evaluation inputs for my ACO

As you can see they all used the same variables although due to ant colony optimization building the starting plan within its code it does not have a workoutPlan input.

As mentioned before the parameters for each algorithm were set after random search testing for them over 50 iterations.

Results

The full results can be found in the appendix.

Algorithm	Time Taken (Seconds)	Most time-consuming function (Total Time)	Function time taken total (Seconds)	Function time taken per call (Seconds)
GA	282.809	fitnessCalc	278.832	1.033
SA	3.110	fitnessCalc	2.748	0.004
ACO	22.734	fitnessCalc	13.226	0.005

Table 30 - Profiling results

The genetic algorithm easily took the most time out of all of them with 283 seconds (rounded), spending 98.5% of that time within the fitnessCalc function. This is due to the large number of individuals in the population (161), contributing to the over 1 second per function call, and the large number of generations (270).

Simulated annealing was the quickest of the three, completing the profiling in just over 3 seconds. It also spent the most time in fitnessCalc, as similarly to the GA it calls fitnessCalc every iteration.

Finally, ant colony optimization took 23 seconds (rounded), this is in the middle of the other two times. One of the bigger differences is the percentage of that time taken up with fitnessCalc being only 58.2%. This is due to it being more complicated outside of the fitness function, having time taken up by heuristic, selectExercise, and updatePheromones functions as they are called every single exercise.

Profiling (With Feedback)

I also performed the same profiling tests with the exact same data other than the user feedback. Here I testing using a mid-length comment which includes one constraint “area” (multiple muscle groups), and a preferred exercise.

The exact quote is “I’ve injured my leg and I want to do more press ups”.

This results in the following output from the LLM:

```
{'constraintExercise': 'Leg Press, Lunges, Bulgarian Split Squats, Jump Squats, Bodyweight Squats, Reverse Lunges w/ DBs, Step-Ups w/ Dumbbells',  
'constraintExerciseSens': '5',  
'preferredExercise': 'Push-Ups',  
'preferredExerciseSens': '3',  
'constraintMuscle': 'Quads, Hamstrings, Calves',  
'constraintMuscleSens': '5',  
'preferredMuscle': 'Chest, Triceps',  
'preferredMuscleSens': '3'}
```

Results

Algorithm	Time Taken (Seconds)	Most time- consuming	Function time taken total (Seconds)	Function time taken per call (Seconds)
-----------	-------------------------	-------------------------	---	--

		function (Total Time)		
GA	191.75	fitnessCalc	186.98	0.69
SA	4.245	fitnessCalc	2.52	0.004
ACO	21.79	fitnessCalc	12.13	0.004

Table 31 - Profiling results with feedback

As you can see compared to the previous results the times of simulated annealing was the only algorithm to take longer with the additional API call, this is of course because of the random starting point and randomness within each algorithm. However, the LLMConstraints() function call did take the most amount of time in a single call of all the function, with 2.19 seconds in the genetic algorithm, 1.41 seconds in simulated annealing, and 1.38 seconds in ant colony optimization.

While this is a significant amount of time for simulated annealing, with roughly 25% extra time, because of the very small total time taken I don't think this would cause for any problems if used in a public application. Due to it only being called once and the total time for genetic algorithm and ant colony optimisation it does not make a significant difference for them.

Time complexity

The time complexity of the genetic algorithm, using Big-O notation is $O(g \times n \times m)$ where:

- g = Number of generations
- n = Population size
- m = Size of individual

For my implementation of it m would be calculated by the session number and session length, as the individual is a single workout plan, made up of a number of days (session number) and a number of exercises per day (calculated from session length).

The time complexity of the simulated annealing algorithm, using Big-O notation is $O(n \times m)$ where:

- n = Number of iterations
- m = Number of exercises

For my implementation m would again have to be worked out by multiplying the session number by the number of exercises as found from the session length.

The time complexity of the ant colony optimisation algorithm, using Big-O notation is $O(n \times m \times k)$ where:

- n = Number of ants
- m = Number of iterations
- k = Number of exercises

Once again, the number of exercises must be calculated as previously described.

Below is a normalized graph showing the time taken by each algorithm as all the parameters in the time complexity algorithms increase, in the dotted lines are the expected lines according to the time complexity. The iterations and size both start at 5 then increase by 5 up until 100 and the session length starts at 20 and goes to 120 increasing by 5 as well, any other parameters that are not shown in the time complexity algorithms are the same as the profiling.

As you can see the lines match close with some differences due to the random nature of the algorithms (using mutation for example).

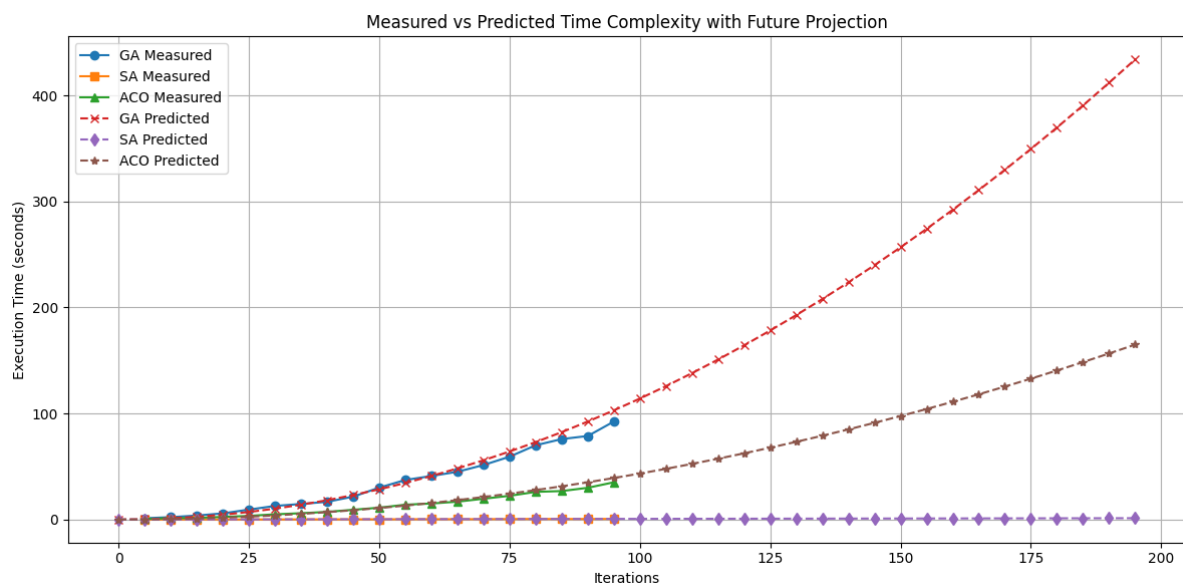


Figure 34 - Graph showing measured vs predicted time complexity

As shown in the graph the genetic algorithm and the ant colony optimization algorithm follow the predicted x^3 line while the simulated annealing algorithm should follow the predicted x^2 line. Due to the size of the numbers the SA line can't be seen well. As expected, the GA takes longer in general hence the steeper curve.

The code that produced the above graph can be found in the graph.py file, it was all generated by ChatGPT and then adjusted by me to fit into my code. I don't take any credit for it.

Evaluation

With the genetic algorithm taking so much longer than the others there is a good argument that the parameters should be adjusted to equalise the field when testing the final output. However, I

will be testing them as they are to keep it consistent and fair, the difference in time taken will be considered in the evaluation.

Overall, the percentage of time the fitness function takes up is not surprising considering the length and amount of logic included within it compared to other functions such as mutation. I believe that due to it being involved in all three algorithms and the driving factor that pushes the workout in the direction that I've decided makes it "good" and effective it is justified to take the time it does. However, I do think it could be optimised further by decreasing the calls to the database by saving it locally in a dictionary before the fitness function is called or process the constraint data outside of the function, so it isn't repeated every time.

The timings are supported by the time complexity for each algorithm, as the genetic algorithm has a large number of generation and population size on every run this will mean it's always going to take a long time even with a much smaller individual size (plan length). As simulated annealing has the most efficient time complexity it is not surprising to see it take the least amount of it by far, it will also not be effected by the size of the population as much as the genetic algorithm and ant colony optimization. Simulated annealing has a similar looking time complexity to the genetic algorithm but the number of ants and iterations is lower than the genetic algorithms generations and population (thanks to the parameter testing), this makes it much quicker.

7 Evaluation

7.1 Introduction

As described in the background and design chapters I will be using ChatGPT model o3-mini as a judge to score the workout plans created, this make it quicker and easier to test many outputs. In order to set a benchmark for this use and specifically my prompt and model choice I have tested 3 workout plans taken from a popular website used for workout plans (Muscle & Strength, n.d.). Below is a table showing the results, the full response with comments can be found in the appendix.

Workout	LLM Difficulty	LLM Focus	LLM Overall
https://www.muscleandstrength.com/workouts/football-strength-workout.html	3	3	3
https://www.muscleandstrength.com/workouts/in-season-workout-routine-for-basketball-players	2	2	2
https://www.muscleandstrength.com/workouts/off-season-baseball-strength-workout.html	3	2	3

Table 32 - Example workouts from internet

As you can see the scores are fairly low, now this should be seen as a benchmark to compare the scores my program generates and not an evaluation of the LLM as due to the limitations of my program it does not account for the extra information given by the web page, and I have not picked the plans based on their effectiveness but on their popularity.

Therefore, if my scores can match or beat these scores I will take that as a sign that my program can create plans to a similar level to the most popular free sport-specific plans currently out there.

7.2 Output testing/LLM-as-a-judge

Set-up

To test the output of my algorithms I set up a function that runs all three algorithms on different threads at the same time, this is to speed it up as it will only take as long as the longest algorithm (Generally the genetic algorithm). The function runs all three algorithms on the same person data, randomly picked once per loop.

Before running the test, I cleared the people table in my database of all data, as it had collected some from my testing of the GUI, I then used ChatGPT to generate code that would create 100

new people using random variables picked within the values that were allowed through my validation function, this can be seen below.

```
# Function to generate 100 unique people
def generate_people(num_people=100):
    people = set()
    while len(people) < num_people:
        name = f"{random.choice(first_names)} {random.choice(last_names)}"
        age = random.randint(18, 85)
        skill = random.randint(1, 5)
        sessions = random.randint(2, 14)
        session_length = random.randint(20, 180)
        sport = random.choice(sports)
        people.add((name, age, skill, sessions, session_length, sport))
    return list(people)

people_data = generate_people(100)
```

Figure 35 – Screenshot of ChatGPT generated code for creating random user information

The names were generated by ChatGPT, and the sports are directly from the sports database. I added the new people data into my database and ran a test to make sure they were usable. Finally, I set the random seed to 1 so I could recreate the results if needed and set up a loop to run the testing function 100 times, using try and except in case there was an error.

For the LLM-as-a-judge prompt I used the structure shown in the design section of context, instruction, input data, output format along with the CLEAR framework and OpenAI suggestion to create the prompt, it can be found in my code in the `llmAsAJudge()` function and below are some screenshots showing the structure more clearly.

Context:

```
prompt = f"""
You are an expert in sports and exercise workouts. Analyze the sport-specific gym workout plan provided and evaluate it based on the following categories.
```

Figure 36 – Screenshot of the context part of my prompt

Instruction:

```

### Task:
1. **Difficulty Scoring**
- Score how well the difficulty of the workout plan matches the users skill on a scale of 1 to 5 where 1 is terrible and 5 is amazing.
- Skill level = 1 then the user is not experienced at all, and skill level = 5 then they are very experienced.
- It should be based on the skill level of the user and should be a balanced so not too easy or too difficult.

2. **Sport Focus Scoring**
- Score how applicable to the sport they are training the workout plan is on a scale of 1 to 5 (The same as task 1).
- The workout plan should be balanced by being a good general workout plan but focusing on exercises/muscles that will help with the chosen sport.

3. **Overall Rating**
- Score the workout on a scale of 1 to 5 (The same as task 1).
- Keep in mind the sport and the difficulty score.

4. **Additional Comments**
- Add at least one sentence explaining your score for both difficulty and sport focus.
- Add any changes you would advise if there are any.
- Add a short sentence with any general comments about the workout.

```

Figure 37 - Screenshot of the instruction part of my prompt

Input data:

```

### Workout Plan:
{bestPlan}

### Sport:
{sport}

### Skill Level:
{skill}

```

Figure 38 - Screenshot of the input data part of my prompt

Output format:

```

### Output Format (Valid JSON):
{
  "Difficulty": "<How well the workout plan matched the users skill, scale of 1 - 5>",
  "Sport Focus": "<How well the workout plan matched the sport, scale of 1 - 5>",
  "Overall Rating": "<The overall rating of the workout plan, scale of 1 - 5>",
  "Additional Comments": "<All additional comments>"
}

Only return valid JSON. If a constraint is not mentioned, use an empty string.

```

Figure 39 - Screenshot of the output format part of my prompt

Results

Below is the data pre-processing.

Algorithm	Average Difficulty Match	Average Sport Focus	Average Overall Score	Average Final Fitness	Average Time Taken	Count
GA	2.75	2.51	2.62	0.27	625.98	100
SA	2.72	2.45	2.51	0.38	24.05	100
ACO	2.85	2.63	2.62	0.41	108.90	100

Table 33 - LLM-as-a-judge results (pre-processing)

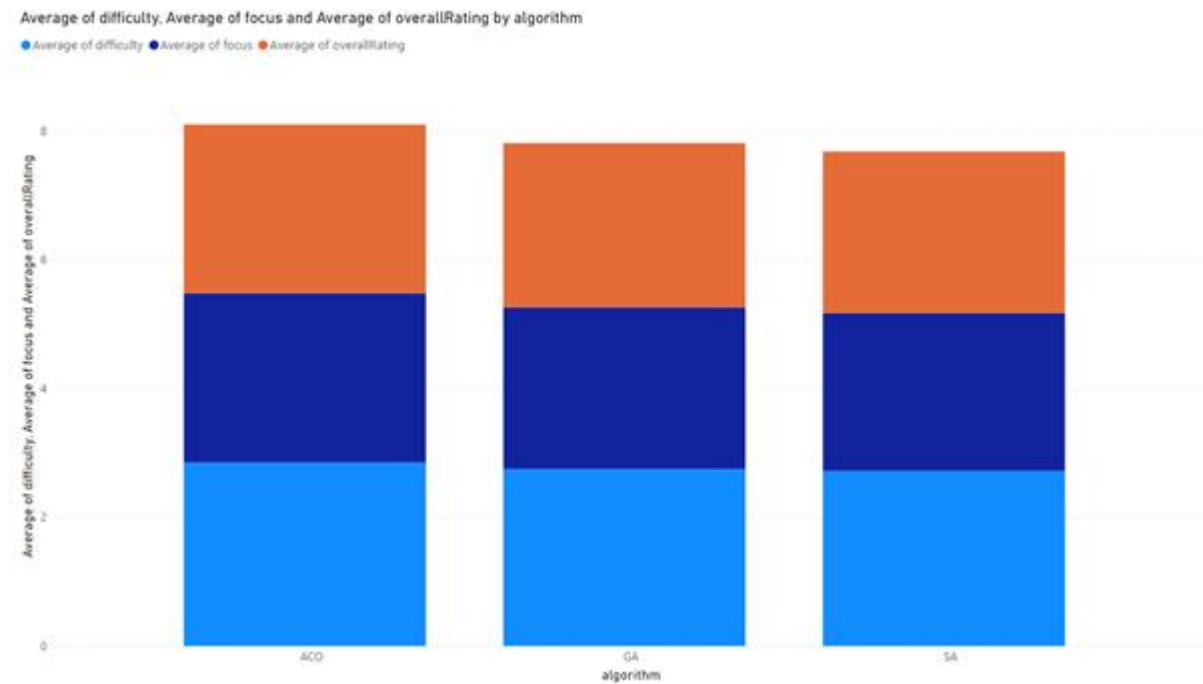


Figure 40 – Stacked bar chart of LLM-as-a-judge data pre-processing

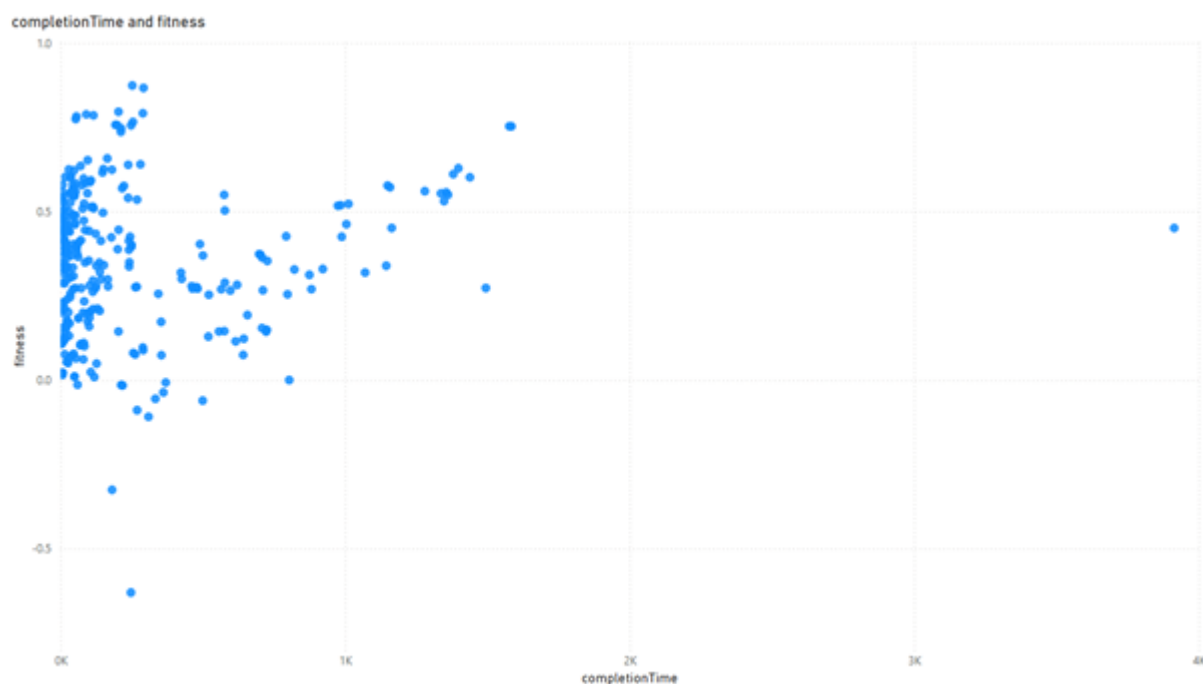


Figure 41 – Scatter graph of LLM-as-a-judge data pre-processing

As you can see there seems to be some outliers and errors produced, these include 12 results which have a minus fitness due to an error in the fitness function (This will be explored in the next section) and one result with a time of 3914 seconds (Rounded up). These have been removed as to better reflect the averages. The following data is once these have been removed.

Algorithm	Average Difficulty Match	Average Sport Focus	Average Overall Score	Average Final Fitness	Average Time Taken	Count
GA	2.85	2.53	2.60	0.32	623.66	88
SA	2.72	2.45	2.51	0.38	24.05	100
ACO	2.87	2.64	2.64	0.41	109.40	99

Table 34 - LLM-as-a-judge results (post-processing)

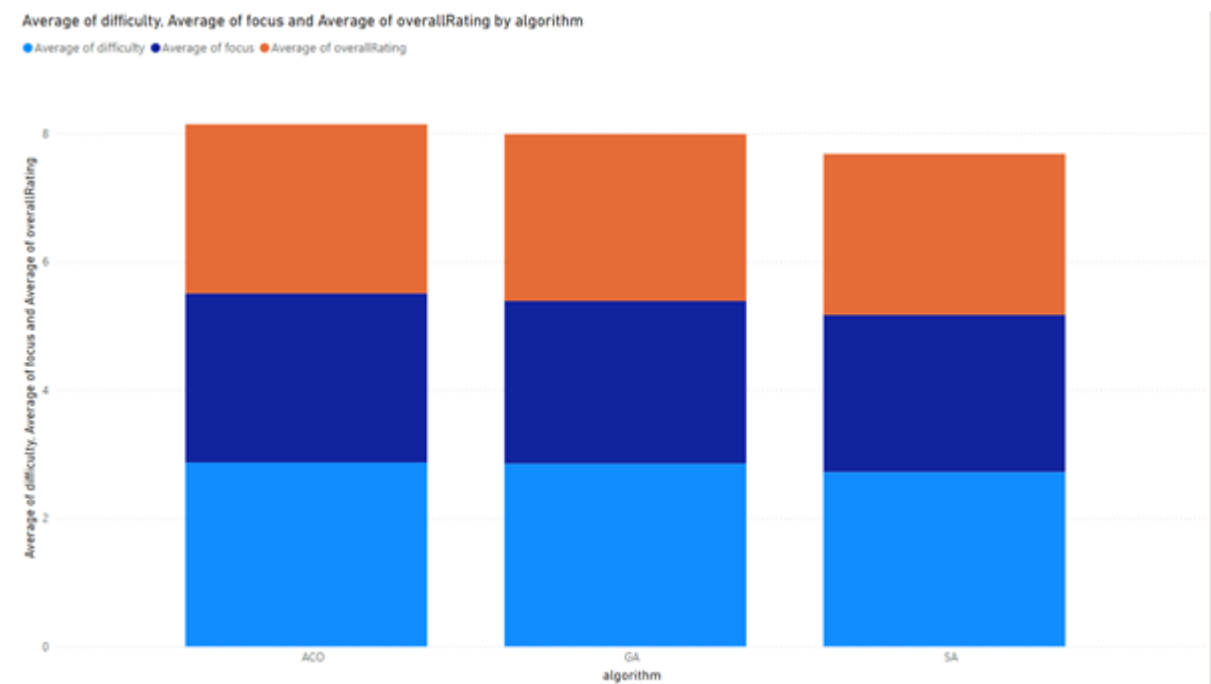


Figure 42 – Stacked bar chart of LLM-as-a-judge data post-processing

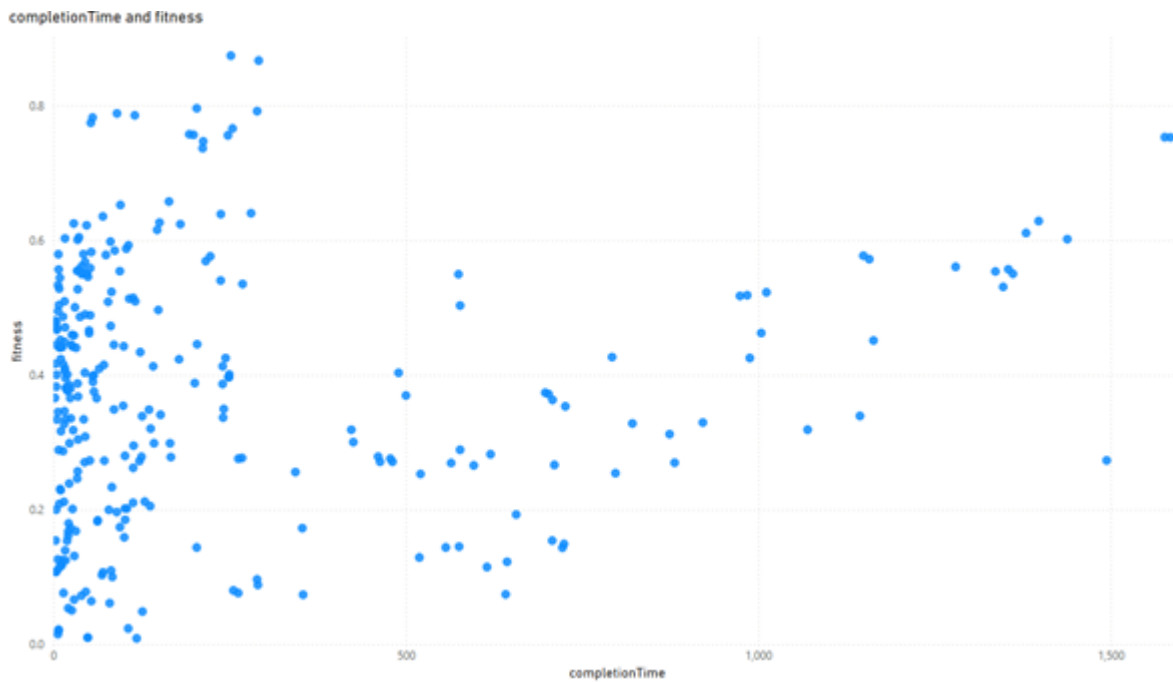


Figure 43 – Scatter graph of LLM-as-a-judge data post-processing

7.3 Evaluation

Across 100 tests the genetic algorithm took the longest average time but produced the lowest average fitness but not the highest average scores from the LLM. Simulated annealing took the shortest time on average and the middle average fitness but the lowest average scores from the LLM, however as seen by the count it didn't produce any results with a negative fitness. Finally, ant colony optimization took a much shorter average time than the genetic algorithm but still significantly longer than simulated annealing, it also scored the highest average fitness, however it did have the highest scores from the LLM and only one error case.

The genetic algorithm while producing good results is held back greatly by the time taken, due to this if I were to use it in the final program the parameters would have to be changed to much smaller numbers, which in turn may lower the LLM scores and the fitness, therefore in this state it would not be my pick in the state it is in.

Simulated annealing would be an ok choice as the time-taken is the closest to an acceptable time for a live-service application, however being the lowest in the LLM scores suggests the pay-off may not be enough. With a result of 2.51 overall it is just above the midpoint, which as described in my prompt to the LLM would make it an ok workout. This would be a good choice for an application that prioritizes speed or maybe a high volume of users.

Ant colony optimization creates better workouts on average according to the LLM scores, despite the highest average fitness score, this suggests my fitness calculation function could be improved to better represent a good workout. Due to this and a balanced time taken this would be my pick for the algorithm to implement into the final production.

Edge Case 1 – Large Session Number/Session Length

Name	Age	Skill	Session Number	Session Length	Sport
Lara Park	47	3	10	120	Surfing

Table 35 - Edge case 1 user

Algorithm	Time-taken	Fitness	Difficulty	Focus	Overall
GA	921.38	0.33	3	3	3
SA	44.67	0.40	3	3	3
ACO	203.88	0.45	3	3	3

Table 36 - Edge case 1 results

Edge Case 2 – Small Session Number/Session Length

Name	Age	Skill	Session Number	Session Length	Sport
Farah Mendez	71	2	2	20	Ice Hockey

Table 37 - Edge case 2 user

Algorithm	Time-taken	Fitness	Difficulty	Focus	Overall
GA	34.48	0.26	2	3	3
SA	2.10	0.37	2	3	2
ACO	7.53	0.29	3	4	3

Table 38 - Edge case 2 results

From these two edge cases we can see that with the increase in workout plan size the fitness and the time taken increase, this backs up the time complexity from the previous section as both the genetic algorithm and the ant colony simulation seem to increase significantly more compared to the simulated annealing when the individual size is increased.

We can also see the scores for the smaller plan are similar but slightly lower in general, I believe this is due to the lower skill level in edge case 2, as this will lessen the amount of exercises that are available to the algorithms.

7.4 Comment analysis

The comments from the LLM can be broken up into 4 different sections, the difficulty match, the sport focus, any changes it would make, and any other general comments. This was specified in the prompt:

“4. ****Additional Comments****

- Add at least one sentence explaining your score for both difficulty and sport focus.
- Add any changes you would advise if there are any.
- Add a short sentence with any general comments about the workout.”

For my analysis I choose to do a thematic analysis of just the general comments, this is because the comments about the difficulty and sport focus generally reflect the score, adding only some explanation on if it's too easy or too difficult and some examples of exercises which cause this. The general comments give more detail which can't be gotten from a score alone as easily. I picked a random sample of 20 comments for each algorithm.

I am going to be using the post-processing data, so without the plans with a negative fitness and the single plan which took over an hour to complete as I feel these outliers skew the data.

Data

Theme	Frequency	Description
More targeted exercises	21	Asking for more sport specific movements.
Diverse	11	Lots of different exercises used (Positive).
Simplify	16	Workout could be simplified.
Too much strength	10	More aerobic exercises needed.
Balanced	5	Good balanced workout

Table 39 - Comment analysis results

Analysis

Overall, the comments seemed to be very focused on the sport match, specifically exercises which mimicked the movements/skills of the chosen sport. This could be due to a lack of exercises in my database as many of the suggested exercises are not included, it also could be changed by specifying within the prompt that these are supposed to be additional workouts on top of regular training.

Many of the workouts with higher ratings mentioned the diverse set of exercises included, while I do still believe adding more exercises to my database would be a good improvement, I think the logic in my fitness function that stops duplicates seems to be working well and encouraging the correct action.

Lots of the lower scored plans had a comment mentioning simplifying them. This was particularly prominent in lower skill level plans. I think this is a valid point as the difficulty of each exercise is taken into account my fitness does not take into account the difficulty of the plan as a whole, this could be a worthwhile addition.

Many plans emphasized too many strength exercises and not enough cardio, this was of course more prominent in sports like long-distance running. I think is another case of my exercise database not including enough cardio type exercises, mainly due to it being more gym focussed but it's an improvement that could be made easily.

Finally, most of the higher scored workouts (4 and above) praised the balance of it, which is like the diverse comments. This seemed to come up on more of the gym-focused sports such as weightlifting. This makes sense due to the previously mentioned focus on gym exercises in the database and the goal of the program which maybe could have been made slightly more clear within the prompt.

Summary

When looking through the comments I noticed that for each algorithm in the same set (Same user information) the comments received were very similar. This leads me to conclude that the algorithms were working well but my fitness function or database could be improved upon. While this doesn't surprise me as it's a very complicated topic to try and reduce into numbers I do think I could have maybe included some additional information in the prompt to specify that this is meant to be additional gym training on top of sport training.

Another interesting point I noticed was the LLM's tendency to only mention the "negative" points in the comments, as the higher scored workouts often had much shorter comment sections and sometimes no additional comments (On top of the explanation of scores) at all.

7.5 Errors

First I want to address the negative fitness values, this is a problem as it causes some of the logic in the fitness function to not work as well, as multiplying a positive number by a percentage such as 0.5 (50%) will cause it to lower however doing the same to a negative number will increase it. Now this could be fixed by ensuring the fitness is being checked by closest to 0 however in much of my code it is chosen based on the lower fitness. I believe the only place in my code which could cause the fitness to become negative is the logic that checks for muscle groups within a single day, this is as follows: $\text{fitness}[\text{day}] *= (1 - (\text{highestCount} / 10))$. If the "highestCount" (Which is the count of the muscle group most repeated in the day) was 11 or above then it would become $1 - 1.1$ which equals -0.1. This is supported by the fact that all of the plans with a negative fitness had large session lengths (119 – 172), making it possible to have more than 10 exercises in a single day. To fix this I will cap highestCount to a max of 9 as with 10 it would make the fitness 0 and any higher it will push it into the negatives, with a max of 9 it still encourages working the same muscle groups on the same day but won't create a negative fitness.

8 Conclusions

8.1 Aim

My aim was to create a program that can output a sport specific workout plan based on user inputs, and adjust it based on user comments. Overall I think I have satisfied my aim, my program does create a sport specific workout plan based on the users skill, sport, time availability, and comments. While there are lots of improvements I would like to make (Discussed in the future work section) I am happy with what has been completed considering the time constraints of a relatively short project.

8.2 Objectives

Objective	Complete?	Chapter
1	Pass	Background – Existing Approaches
2	Pass	Background - Abstraction
3	Pass	Background – WCSP solutions
4	Pass	Background – Supporting Code
5	Pass	Methodology/Design
6	Pass	Design
7	Pass	Testing
8	Pass	Testing/Evaluation/Conclusion

Table 40 - Conclusion objectives

Objective 1,2, 3, and 4 were all met in the background chapter, these were looking at the existing approaches and how they succeed and fail, how my problem can be seen as a mathematical problem, solutions to that problem backed up by previous research, and the use of an LLM to parse user comments. I think that within my background section I have covered all of the requirements for my objectives with the proper supporting research, this helped me not only choose what I will be using within my code, but also how I should plan and design my project as a whole.

Objective 5 described having a plan for my project. While I do think I have passed as I did use checklists and timelines to keep myself on track, as seen in the methodology chapter, I do think that the way I managed my project would not be suitable for a larger project or larger team. For my use of a single-person, roughly 6 month long project it allowed me to stay flexible while still keeping on track.

Objective 6 covered the software design of my project, specifically the code and data storage/flows. I used tables, flowcharts, and wireframes to design my back-end code, my database, and my front-end GUI.

Objective 7 described the testing I should complete. I covered this in my testing chapter where I used unit testing, profiling, and time-complexity to ensure my code worked fully and as expected without crashes or errors, and to identify any failures or bottlenecks.

Finally, objective 8 describes evaluating my algorithms to find the best solution using time-complexity and statistics. As mentioned in my last paragraph I have used time-complexity and statistics based on the LLM-as-a-judge scores, fitness, and time taken to complete a full evaluation of my programs outputs.

8.3 Results of testing/evaluation

As seen in my testing chapter my code works fully and correctly in almost all tests, it does have some bottlenecks and hold ups, specifically the fitness function taken up such a large amount of the runtime but it does not crash and only presented a few errors (The negative fitness' from the evaluation chapter, fix described there), unfortunately the errors were found through running the algorithms with random users and not my software testing, this is a problem because it shows I missed it.

In my evaluation chapter I mainly discussed the LLM-as-a-judge scores, these showed that despite on average not being as high as I would like, they do match the small sample of popular workout plans I found on the internet, and therefore I would conclude that my program is successful. If I were to fully launch this program as an application I would most likely use the ant colony optimization algorithm. This has the best scores from the LLM on average, took an acceptable amount of time depending on the individual size, and generally ended with a respectable final fitness. I think it has the correct balance for use in an app where the user is not going to want to wait more and a minute or two for an algorithm to be generated and it can be edited through the parameters to be faster or have better results (Perhaps a feature users could choose). The genetic algorithm was the algorithm I expected to be best when I first started, but due to the amount of time it takes to complete I don't think it would be suitable for real users, with some tuning of the parameters and perhaps making the fitness function more efficient I do think it could work well. Simulated annealing was easily the quickest and least computationally complex algorithm, making it a great option for quick fire plans which may not be the best ever but certainly better than no workout plan at all.

In a real life application of my program I believe the best use would be a mix of all three algorithms, based on a choice by the user on what they prioritise.

8.4 Future Work

As previously mentioned, I do think I have met my aim with my program, however there are lots of improvements I would make if I had the time. Firstly my fitness function could be improved in a couple of ways, I would like to optimize it to make it more efficient so that the time taken is shortened, I would also like to add to it so that it could be more precise, I haven't not used the users age for example. Another area that needs improving is the data, I don't think there are enough exercises in my database, this would improve lots of the issues around making a plan for a user with lots of exercises or a low skill level. For the program as a whole I think adding a system to calculate optimal reps, sets, and RPE (Rate of perceived exertion, used to suggest weight of the exercise without explicitly giving a weight) would be useful for users with no previous experience. Finally, I would definitely put more time into fine-tuning the parameters for each algorithm, the prompt used for judging the workout plan, and the penalties in the fitness function.

References

1. Agrawal, P., et al. (2023). Impact of Prompt Engineering on Clinical LLM Performance. *Journal of Biomedical Informatics*, 144. Available at: <https://pmc.ncbi.nlm.nih.gov/articles/PMC11036183/#sec15>.
2. Baeldung. (2024). *Branch and Bound Algorithm*. Available at: <https://www.baeldung.com/cs/branch-and-bound>.
3. Bender, E.M., Gebru, T., McMillan-Major, A. and Shmitchell, S. (2021). On the Dangers of Stochastic Parrots. *FAccT '21*, pp.610–623.
4. British American Football, (2021). *BAFA Annual Report*. [online] Available at: <https://www.britishamericanfootball.org/wp-content/uploads/2021/11/BAFA-Annual-Report-2021-1.pdf>.
5. Chen, C.H., et al. (2009). Dynamic Penalty Methods for Genetic Algorithms. *IEEE Transactions on Systems, Man, and Cybernetics*, 39(1), pp.1–12.
6. ChIBM. (2024). *Constraint Propagation – IBM ILOG CPLEX*. Available at: <https://www.ibm.com/docs/en/icos/22.1.0?topic=optimizer-constraint-propagation>.
7. Christophe Lecoutre (2013). *Constraint Networks*. John Wiley & Sons.
8. customtkinter Documentation (n.d.). *CustomTkinter Widgets*. Available at: <https://customtkinter.tomschimansky.com/documentation/widgets/checkbox>
9. DataCamp (n.d.). *Genetic Algorithm in Python: A Practical Guide*. Available at: <https://www.datacamp.com/tutorial/genetic-algorithm-python>
10. Dorigo, M. (1992). *Optimization, Learning and Natural Algorithms*. PhD thesis, Politecnico di Milano.
11. FitnessAI. (2024). *FitnessAI – Personalized Workout Plans*. Available at: <https://www.fitnessai.com/>.
12. Floudas, C.A. and Pardalos, P.M. (2009). *Encyclopedia of optimization. Vol. 5 : O-R*. New York: Springer.
13. Gambardella, L.M. and Dorigo, M. (2007). Ant colony system for job-shop scheduling. *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, 31(1), pp.70–77. Available at: <https://ieeexplore.ieee.org/.../document/4129846>.
14. GeeksforGeeks. (2024a). *Local Search Algorithm in Artificial Intelligence*. Available at: <https://www.geeksforgeeks.org/local-search-algorithm-in-artificial-intelligence/>.
15. GeeksforGeeks. (2024b). *Introduction to Ant Colony Optimization*. Available at: <https://www.geeksforgeeks.org/introduction-to-ant-colony-optimization/>

16. GOV.UK. (2016). *Dos and don'ts on designing for accessibility*. Available at: <https://accessibility.blog.gov.uk/2016/09/02/dos-and-donts-on-designing-for-accessibility/>.
17. Government Digital Service. (2016). *Dos and don'ts on designing for accessibility*. Available at: <https://accessibility.blog.gov.uk/2016/09/02/dos-and-donts-on-designing-for-accessibility/>
18. Granacher, U. and Borde, R. (2017). Effects of Sport-Specific Training during the Early Stages of Long-Term Athlete Development on Physical Fitness, Body Composition, Cognitive, and Academic Performances. [online] Frontiers. Available at: <https://www.frontiersin.org/journals/physiology/articles/10.3389/fphys.2017.00810/full>.
19. Huang, X., Liu, Y., Zhang, Y. et al. (2024). A Survey on LLM-as-a-Judge. *arXiv preprint*. Available at: <https://arxiv.org/pdf/2411.15594>.
20. Jefit. (2024). *Barbell Zercher Squat Exercise Guide*. Available at: <https://www.jefit.com/exercises/219/barbell-zercher-squat>
21. Mental Health Foundation (2025). *How to look after your mental health using exercise*. [online] Mental Health Foundation. Available at: <https://www.mentalhealth.org.uk/explore-mental-health/publications/how-look-after-your-mental-health-using-exercise>.
22. Michalewicz, Z. and Schoenauer, M. (1996). Evolutionary algorithms for constrained parameter optimization problems. *Evolutionary Computation*, 4(1), pp.1–32.
23. Microsoft. (1996). *Verdana font family overview*. Available at: <https://learn.microsoft.com/en-us/typography/font-list/verdana>
24. Mixpaper. (2023). *Prompt Engineering Definition*. Available at: https://file.mixpaper.cn/paper_store/2023/681177f8-cd15-4e0f-a23b-997c6b9f9dd2.pdf.
25. Muscle & Strength (n.d.) *Muscle & Strength*. Available at: <https://www.muscleandstrength.com/>.
26. Nichols, G. (2014). *Sport and Crime Reduction : the Role of Sports in Tackling Youth Crime*. Hoboken: Taylor and Francis.
27. OpenAI (2024). *Quickstart Guide for API Integration*. Available at: <https://platform.openai.com/docs/quickstart?api-mode=chat>
28. OpenAI. (2024). *Prompt Engineering Guide*. Available at: <https://platform.openai.com/docs/guides/prompt-engineering>.
29. OpenAI. (2024). *Prompt Engineering Guide*. Available at: <https://platform.openai.com/docs/guides/prompt-engineering>

30. Ormerod, C., Malhotra, S. and Jafari, A. (2021). Automated Essay Scoring Using Efficient Transformer-Based Language Models. *arXiv preprint*. Available at:
<https://arxiv.org/pdf/2102.13136>.
31. Ramezani, M. and Babamir, S.M. (2005). A simulated annealing-based approach to solving constraint satisfaction problems. *Reliability Engineering & System Safety*, 90(1), pp.123–130.
32. ResearchGate. (2011). *Topology Optimization for Next Generation Access Networks: Algorithmic Overview*. Available at:
https://www.researchgate.net/publication/221706895_Topology_Optimization_for_Next_Generation_Access_Networks_Algorithmic_Overview
33. ResearchGate. (2020). *Benchmarking Metaheuristic Algorithms for Capacitated Coverage Path Planning Problems*. Available at:
https://www.researchgate.net/publication/344510683_An_Arable_Field_for_Benchmarking_of_Metaheuristic_Algorithms_for_Capacitated_Coverage_Path_Planning_Problems
34. ResearchGate. (2020). *Genetic Algorithm Based on Natural Selection Theory for Optimization Problems*. Available at:
https://www.researchgate.net/publication/344832850_Genetic_Algorithm_Based_on_Natural_Selection_Theory_for_Optimization_Problems
35. Reynolds, L. and McDonell, K. (2021). Prompt engineering techniques for large language models. *IEEE Transactions on Neural Networks and Learning Systems*, 32(11), pp.4899–4913.
36. Schiex, T., Fargier, H., & Verfaillie, G. (1998). Valued constraint satisfaction problems: Hard and easy problems. *European Journal of Operational Research*, 130(2), pp. 238–255.
37. Schoenfeld, B. J., Ogborn, D., & Krieger, J. W. (2021). *Effects of resistance training frequency on measures of muscle hypertrophy: A systematic review and meta-analysis*. *Sports*, 9(3), 32. Available at:
<https://www.ncbi.nlm.nih.gov/pmc/articles/PMC7927075/>
38. Schoenfeld, B. J., Ratamess, N. A., Peterson, M. D., Contreras, B., Tiryaki-Sonmez, G., & Alvar, B. A. (2014). *Effects of different volume-equated resistance training loading strategies on muscular adaptations in well-trained men*. *Journal of Strength and Conditioning Research*, 28(10), pp.2909–2918. Available at:
https://journals.lww.com/nsca-jscr/fulltext/2014/10000/effects_of_different_volume_equated_resistance.27.aspx

39. SISL. (n.d). *Introduction to Optimization*. Available at:
<https://web.stanford.edu/group/sisl/k12/optimization/MO-unit1-pdfs/1.1optimization.pdf>.
40. SMU. (2006). *Weighted CSPs in Real Applications*. Available at:
https://ink.library.smu.edu.sg/cgi/viewcontent.cgi?article=1075&context=sis_research.
41. Sommerville, I. (2011). *Software Engineering*. 9th ed. Boston: Addison-Wesley. Available at: <https://ieeexplore.ieee.org/abstract/document/6068383>.
42. Sportengland.org. (2020). Active Lives | Results. [online] Available at:
<https://activelives.sportengland.org/Result?queryId=277> [Accessed 31 Oct. 2024].
43. Sportengland.org. (2020). *Active Lives | Results*. [online] Available at:
<https://activelives.sportengland.org/Result?queryId=277> [Accessed 31 Oct. 2024].
44. Springer. (2023). *CLEAR Framework for Prompt Engineering*. *Computers in Biology and Medicine*, 151, 106330. Available at:
<https://www.sciencedirect.com/science/article/pii/S0099133323000599>
45. Springer. (2023a). Prompt Engineering in Biomedical Applications. *Annals of Biomedical Engineering*, 51(1), pp.34–48.
46. Springer. (2023b). *CLEAR Framework for Prompt Engineering*. *Computers in Biology and Medicine*, 151, 106330.
47. Stack Overflow. (2024). *Developer Survey 2024*. Available at:
<https://survey.stackoverflow.co/2024>
48. Stanislav Živný (2012). The Complexity of Valued Constraint Satisfaction Problems.
49. UK Coaching Population study 2022 England -Report Coaching in the UK 2022 All reports are produced by and edited by UK Coaching 2. (n.d.). Available at:
<https://www.ukcoaching.org/UKCoaching/media/coaching-images/Entity%20base/Downloadables/UK-Coaching-population-study-2022-England.pdf>.
50. Ukcoaching.org. (2020). UK Coaching Population study 2022 England -Report Coaching in the UK 2022 All reports are produced by and edited by UK Coaching 2. (n.d.). Available at: <https://www.ukcoaching.org/UKCoaching/media/coaching-images/Entity%20base/Downloadables/UK-Coaching-population-study-2022-England.pdf>.
51. Wilkerson, J.W. and Tauritz, D.R. (2010). Outlining a practitioner’s guide to fitness function design. *GECCO Proceedings*, pp.2719–2726.
52. www.facebook.com/thefitnessgrp (2024). How Much is a Personal Trainer? Latest 2023 Guidance. [online] Thefitnessgrp.co.uk. Available at:
<https://www.thefitnessgrp.co.uk/how-much-is-a-personal->

- [trainer/#How Much is a Personal Trainer Regional Costs Revealed](#) [Accessed 1 Nov. 2024].
53. www.facebook.com/thefitnessgrp (2024). *How Much is a Personal Trainer? Latest 2023 Guidance*. [online] *Thefitnessgrp.co.uk*. Available at: <https://www.thefitnessgrp.co.uk/how-much-is-a-personal-trainer/#How Much is a Personal Trainer Regional Costs Revealed> [Accessed 1 Nov. 2024].
54. Zimmermann, H.J. (1984). Fuzzy programming and linear programming with fuzzy coefficients. *European Journal of Operational Research*, 16(1), pp.86–95. Available at: <https://link.springer.com/article/10.1007/BF01759923>.

Appendix A Personal Reflection

A.1 Reflection on Project

Sport science and how to make a good workout is itself a topic that many papers have been written about and will continue to be written about, due to this my project has an almost endless number of possible improvements. For example, the reps/sets and more specific structured components to the workout were not in my project scope, however I would have liked to add them if I had more time.

Another problem I encountered was the testing of these workouts, within the dissertation I have used large language models to judge the output of each algorithm, while I think this works well and was very helpful in teaching me important skills such as prompt engineering, in a perfect world I would have liked to use real experts. This would have been difficult as I would need them to be an expert on gym workouts as well as a specific sport, which can be difficult to find and much more difficult to get in touch with.

My time management for my project was good for the most part although I do think putting more work in before the winter break would have allowed for a more relaxed and possibly better outcome.

Throughout the project I used ChatGPT for help, this mainly focused on structuring the different chapters, brainstorming, and some code. This has all been mentioned within my dissertation and any code has been commented on as well. I did find it useful but very quickly found the flaws in even the latest models, for example it would often miss the point of a chapter and I would have to specify particular sections and the order I was going to use, because of this it was used similarly to how talking a problem through can help, in that the replies are not the useful part but forcing myself to break it down to explain it better helped me process it better.

A.2 Personal Reflection

I am overall happy with the outcome of my project, I put in a lot of time and effort, and I think it is shown in my dissertation. In particular I am proud of my code as it is the first time I've used python other than in the AI module of this year and it's a language that I have wanted to improve on for a long time, I think my implementation and all the debugging have pushed my coding to a better level than in previous years.

One part of the project I found particularly difficult was the background section/literature review, I find that a lot of the choices I make are due to them being the easiest way or the most "common sense" method. This of course is not enough for a final year project and so I had to constantly check myself and make sure what I am doing makes sense and can be backed up by

literature. I still think I have lots to improve on in this regard and if I were to continue to study it would be a skill I would have to practice a lot.

The time I spent at Eli Lilly on my placement helped me out a lot throughout my project, firstly the 9-5 work schedule and ethos was really put to use from January as the amount of work combined with the difficulty of some of it required lots of time and well-spaced breaks. I used a timer to motivate myself to do stints of 45 minutes with a 15 minutes break between to keep myself working hard over many hours.

I hope that some of the skills I learnt through the project can be added to my CV and help me in securing future jobs, for example my Python knowledge is potentially very helpful in the world of companies trying to use AI effectively, additionally the knowledge I've gained about optimization through my research can be highlighted and hopefully used to speed up existing infrastructure at a company.

Overall I am proud of my work and hope to use it within my CV to boost my employability and success.

Appendix B Ethics Documentation

B.1 Ethics Pro-forma

30/01/2025, 13:30

Research Ethics Supervisor Pro forma



Research Ethics Supervisor Pro forma

This form should be completed in collaboration with your supervisee. Please ensure you have familiarised yourself with the University Research Ethics Process prior to supporting students. For more information please see the Research Ethics and Integrity webpages. For further information and guidance, please contact the central Research Ethics team via email to res-ethics@brunel.ac.uk

Supervisor



Please complete the following:

1. Student First Name *

Caspar

2. Student Surname *

Ashworth

3. Student Number *

2010518

4. Module Name *

<https://forms.office.com/Pages/ResponseDetailPage.aspx?id=sZetTDVZA0GoZlemKFRfvDEfUQaCxRRFtblcvHzJTVUMkFXWkZBTkRERjFWQINZUjlaQU...> 1/4


30/01/2025, 13:30

Research Ethics Supervisor Pro forma

Computer Science Final-Year Project


5. Module Code * 

CS3072


6. Submission deadline * 

11/4/2025



7. Please provide the BREO reference ID for the approved Module Application relevant to this student's research proposal (if applicable): * 

48594-MA-Oct/2024-52961-3

8. Please provide a brief description of the proposed low risk research (including research method(s), intended participants and recruitment method) * 

My project will be using and comparing hybrid algorithms, these will be made up of a weighted constraint satisfaction solution (such as local search or a genetic algorithm) and a solution to the multi-object optimization problem (such as a genetic algorithm). This will be used in the context of a sport-specific workout plan, based on user inputs such as sport, time availability, and proficiency. I will assess it with the Big-O notation and time to complete, as well as the success of each individual algorithm by the fitness/weight of the final solution. I will use a database of exercises taken from a public database and saved to my local device to make changes such as adding variables which will be weighted so they can be

9. Are you satisfied the proposed research falls under BUL's stated risk categories and the parameters of the approved Module Application?

You can view the Research ethics risk categories here -

<https://www.staff.brunel.ac.uk/research-ethics-risk-categories>

* 

30/01/2025, 13:30

Research Ethics Supervisor Pro forma

☒ Yes

☐ No

Supervisor Confirmation



Please read the following statement

- I confirm I have met with the student and discussed the research proposal in full.
- I confirm that I consider the proposed research is low-risk in nature and permissible under the parameters of the approved Module Application.
- I confirm that I have advised the student on the ethical aspects of the study design and their responsibilities when conducting research.
- I confirm that the student has been advised to read the University's Code of Research Ethics, the BUL research Risk Categories and other relevant documentation.
- I confirm that the student has the skills to carry out the research.
- I confirm that if the research involves human participants, the necessary documentation (Participant Information Sheet, consent forms etc.) have been checked and completed appropriately.
- I confirm that if the research involves human participants, the procedures for recruitment and obtaining informed consent are appropriate.

10. Please provide your Brunel email below to confirm you have read and agree to the above conditions *

philipp.bibik@brunel.ac.uk

Student Declaration (to be completed by the supervisor on behalf of the student)



Please review the following statements

- I confirm I have met with my supervisor and discussed my research proposal in full.
- I confirm that the research will be undertaken in accordance with the Brunel University London Code of Research Ethics and the Brunel University London Research Integrity Code.
- I confirm that my proposed research is low-risk in nature according to the risk categories set by the University.
- I agree to notify my supervisor before implementing any changes to the agreed protocol/methods.

<https://forms.office.com/Pages/ResponseDetailPage.aspx?id=sZetTDVZA0GoZletmKFRfvDEfUQaCxRfTablcVHzJTIVUMkFXWkZBTkRERjFWQINZUjlaQU...> 3/4

30/01/2025, 13:30

Research Ethics Supervisor Pro forma

- I understand that if at a later date I decide to change my research study, this confirmation may no longer be valid.
- I understand that I may conduct my study only as agreed with my supervisor, and that any research activity of a medium or high-risk nature without the necessary approval may result in disciplinary procedures.

11. Please provide your Brunel email below to confirm your student has read and agreed to the above conditions *

2010518@brunel.ac.uk



This content is created by the owner of the form. The data you submit will be sent to the form owner. Microsoft is not responsible for the privacy or security practices of its customers, including those of this form owner. Never give out your password.

Microsoft Forms | AI-Powered surveys, quizzes and polls [Create my own form](#)

[Privacy and cookies](#) | [Terms of use](#)

Appendix C Video Demonstration

C.1 Link to the video

<https://youtu.be/HwgPNBeuX1Y>

Appendix D Other Appendices

More relevant material

Profiling results

Tue Mar 25 12:49:31 2025 profileResults

28066580 function calls (28066540 primitive calls) in 282.809 seconds

Ordered by: cumulative time

List reduced from 63 to 10 due to restriction <10>

ncalls	tottime	percall	cumtime	percall	filename:lineno(function)
1	0.000	0.000	282.809	282.809	{built-in method builtins.exec}
1	0.000	0.000	282.809	282.809	<string>:1(<module>)
1	0.000	0.000	282.809	282.809	parameterTestMain.py:160(runProfileGA)
1	0.144	0.144	282.809	282.809	main.py:663(geneticAlgorithm)
270	15.371	0.057	278.832	1.033	main.py:267(fitnessCalc)
3543993	216.768	0.000	216.768	0.000	{method 'execute' of 'sqlite3.Cursor' objects}
3543129	37.312	0.000	37.312	0.000	{method 'fetchone' of 'sqlite3.Cursor' objects}
174956	0.727	0.000	2.414	0.000	__init__.py:599(__init__)
174956	0.272	0.000	1.687	0.000	__init__.py:673(update)
218695	0.575	0.000	1.640	0.000	{built-in method builtins.sum}

Tue Mar 25 12:49:34 2025 profileResults

397449 function calls in 3.110 seconds

Ordered by: cumulative time

List reduced from 55 to 10 due to restriction <10>

ncalls	tottime	percall	cumtime	percall	filename:lineno(function)
1	0.000	0.000	3.110	3.110	{built-in method builtins.exec}
1	0.000	0.000	3.110	3.110	<string>:1(<module>)
1	0.000	0.000	3.110	3.110	parameterTestMain.py:178(runProfileSA)
1	0.012	0.012	3.110	3.110	main.py:916(simulatedAnnealing)
622	0.142	0.000	2.748	0.004	main.py:267(fitnessCalc)
52250	2.185	0.000	2.185	0.000	{method 'execute' of 'sqlite3.Cursor' objects}
621	0.019	0.000	0.347	0.001	main.py:577(mutation)

```

51004    0.338    0.000    0.338    0.000 {method 'fetchone' of 'sqlite3.Cursor'
objects}
1246     0.198    0.000    0.219    0.000 {built-in method _sqlite3.connect}
1246     0.053    0.000    0.053    0.000 {method 'fetchall' of 'sqlite3.Cursor'
objects}

```

Tue Mar 25 12:49:57 2025 profileResults

11882991 function calls (11882834 primitive calls) in 22.734 seconds

Ordered by: cumulative time

List reduced from 204 to 10 due to restriction <10>

```

ncalls  tottime  percall  cumtime  percall filename:lineno(function)
5/1     0.000    0.000    22.734    22.734 {built-in method builtins.exec}
1       0.000    0.000    22.734    22.734 parameterTestMain.py:196(runProfileACO)
1       0.216    0.216    22.734    22.734 main.py:820(antColonyOptimization)
2886    0.700    0.000    13.226    0.005 main.py:267(fitnessCalc)
236817  9.808    0.000    9.808    0.000 {method 'execute' of 'sqlite3.Cursor'
objects}
77922   7.361    0.000    8.875    0.000 main.py:756(selectExercise)
236812  1.561    0.000    1.561    0.000 {method 'fetchone' of 'sqlite3.Cursor'
objects}
9780789 1.342    0.000    1.342    0.000 {method 'append' of 'list' objects}
3051    0.629    0.000    0.691    0.000 {built-in method _sqlite3.connect}
156     0.307    0.002    0.308    0.002 main.py:791(updatePheromones)

```

Full results for online workouts

Workout	LLM Response
https://www.muscleandstrength.com/workouts/football-strength-workout.html	{'Difficulty': '3', 'Sport Focus': '3', 'Overall Rating': '3', 'Additional Comments': 'For an intermediate (skill level 3) athlete, the exercise choices are challenging enough without being overwhelming, though some exercises like wide-grip pull-ups may require solid form. The plan builds general strength that is beneficial for American football, but it lacks sport-specific elements such as plyometric, agility, and explosive power drills. Consider incorporating these elements and varying the workout structure to better target the dynamic nature of football.'}
https://www.muscleandstrength.com/workouts/in-season-workout-routine-for-basketball-players	{'Difficulty': '2', 'Sport Focus': '2', 'Overall Rating': '2', 'Additional Comments': 'The workout includes exercises like nordic hamstring curls and cable rows that may be

	too challenging for a beginner (skill level 2), hence the difficulty rate is low. In terms of sport focus, while it does build overall strength, core stability, and lower body engagement, it lacks basketball-specific elements such as plyometric training, agility drills, and vertical jump work. I would advise simplifying or modifying some of the more advanced exercises and adding sport-specific drills to improve both safety and on-court performance. Overall, it's a solid general strength workout that needs adjustments to better match the user's experience and the demands of basketball."}
https://www.muscleandstrength.com/workouts/off-season-baseball-strength-workout.html	{'Difficulty': '3', 'Sport Focus': '2', 'Overall Rating': '3', 'Additional Comments': 'The workout plan offers a moderate level of difficulty that could suit an intermediate (skill level 3) lifter if weights and progressions are adjusted appropriately; however, repeating the same set of exercises over all three days may lead to fatigue or overuse issues. While the exercises provide overall strength building, they are not tailored to the specific needs of baseball, missing key elements such as rotational power, explosiveness, and agility. Adding sport-specific drills (like rotational cable movements, medicine ball throws, plyometrics, and agility drills) would enhance its suitability for baseball performance.'}

Code files explanation

File name	Description
Main.py	This is where the majority of my back-end code is, this includes all the algorithms and some testing.
GUI.py	This is where the code for the GUI is kept, other than the user validation which is in main.
Database.py	This is where the functions I have used to manipulate the database are kept.
UnitTestMain.py	This holds all the unit tests.
ParameterTestMain.py	This is where I tested the different variables for each algorithm, it also holds the profiling functions.
Graph.py	This is only ChatGPT generated code which was used to create the time complexity graph.