

Evolutionary Algorithms for Optimisation Problems: Genetic Algorithms for Portfolio Optimisation

Final Project Report

Author: Yunus Sufian

Supervisor: [redacted]

Student ID: [redacted]

April 4, 2025

Abstract

Rooted within the principles of natural evolution, genetic algorithms embody the power and elegance of evolutionary processes from the dawn of mankind, and this project demonstrates their overlooked utility in solving some of the most expensive problems in the world in an ever growing financial services industry, valued at over $33 trillion today.

We will build three genetic algorithms, each evolving in complexity and utility, to understand how they change based on different hyperparameter setups and adapt to solving problems faced in portfolio optimisation, as well as explore less chartered territory in this research space.

Originality Avowal

I verify that I am the sole author of this report, except where explicitly stated to the contrary. I grant the right to King's College London to make paper and electronic copies of the submitted work for purposes of marking, plagiarism detection and archival, and to upload a copy of the work to Turnitin or another trusted plagiarism detection service. I confirm this report does not exceed 25,000 words.

Yunus Sufian

April 4, 2025

Acknowledgements

I would like to thank my supervisor [redacted] for his insightful guidance, support and encouragement throughout the course of this project.

Contents

1 Introduction **3**

1.1 Problem Definition . . . 4

1.2 Relevance & Motivation . . . 4

1.3 Aims & Objectives . . . 5

2 Literature Review **7**

2.1 Modern Portfolio Theory and Optimisation Challenges . . . 7

2.2 Evolutionary Algorithms . . . 10

2.3 Genetic Algorithms . . . 11

2.4 Optimisation Problems . . . 13

2.5 Key Takeaways and Research Gaps . . . 14

3 Requirements & Specification **16**

3.1 Functional Requirements . . . 16

3.2 Non-Functional Requirements . . . 18

3.3 Optional Requirements . . . 19

3.4 Design Rationale . . . 19

3.5 Formal Specifications . . . 21

4 Methodology & Implementation **24**

4.1 Tooling & Technology . . . 24

4.2 Data Acquisition . . . 27

4.3 Genetic Algorithm Implementation . . . 29

4.4 Testing . . . 47

5 Results, Analysis & Evaluation **49**

5.1 GA1 Results . . . 49

5.2 GA2 Results . . . 53

5.3 GA3 Results . . . 57

5.4 Evaluation vs. Requirements . . . 66

5.5 Comparative Analysis . . . 71

6 Legal, Social, Ethical & Professional Issues **73**

6.1 Legal Issues . . . 73

6.2 Social and Ethical Implications . . . 73

6.3 Professional Issues . . . 74

7 Conclusion **75**
7.1 Summary of Key Findings 75
7.2 Limitations 76
7.3 Future Work 76

References **78**

A User Guide **81**
A.1 Instructions 81

B Source Code **84**

Chapter 1

Introduction

The stock market is a dynamic arena in the modern world, where stocks representing partial ownership in institutional companies are continuously traded almost every second, hour and day. This ongoing activity, marked by rapid fluctuations and evolving trends, highlights the existence of, and need for robust and risk-adjsted investment strategies. In financial terms, a portfolio is a curated collection of these assets, fundamentally designed to balance risk and reward [1] in alignment with an investor's objectives.

Portfolio optimisation is a critical task and a real problem in financial management, in which one aims to allocate valued assets in a manner that generally maximises returns while minimising risk and or other objectives. There are multiple tested and traditional methods to perform portfolio optimisation, one taking the main spotlight being mean-variance optimisation, introduced by the father of Portfolio Theory - Harry Markowitz, who popularised the concept of diversification and essentially proving how the relationship between two assets (low risk, low reward and high risk, high reward) is not linear in terms of risk and return, and instead is a curve known as the efficient frontier.

The challenge lies in developing a flexible and adaptive approach to portfolio optimisation that can move beyond the deterministic boundaries of traditional methods like Mean-Variance Optimisation (MVO). While effective under certain assumptions, classical approaches often struggle with real-world complexities such as non-convex constraints and scalability when dealing with additional objectives like limiting the number of assets in a portfolio (cardinality) for example. This project investigates how evolutionary algorithms, specifically Genetic Algorithms (GA), enhanced with elements of Particle Swarm Optimisation (PSO), can be used to model and solve portfolio optimisation problems under such complex and dynamic conditions,

and by exploring how these algorithms can navigate larger and more irregular search spaces, the project aims to demonstrate their potential as robust alternatives in financial decision-making contexts.

1.1 Problem Definition

The core problem tackled here is selecting an optimal portfolio from a finite pool of stocks under multiple, often conflicting objectives, such as balancing returns and risk while considering constraints like cardinality (the number of assets held). For instance, consider a scenario where one must choose 10 stocks for a portfolio, out of a pool of 100 due to capital restrictions, transaction limits or portfolio rebalancing caps. Capital restrictions can often arise from limited available funds, minimum trade sizes, or regulatory requirements - which all prevent investors from allocating small amounts of capital across a large number of assets. This limits cardinality in the portfolio by making it impractical or inefficient to hold too many positions, especially when accounting for transaction costs and liquidity constraints. In this situation we would have to search for combinations that satisfy these flexible, soft constraints; balancing overall portfolio composition without imposing rigid limits on individual stocks as hard constraints.

Traditional methods, like Mean Variance Optimisation (MVO), struggle when the solution space is non-convex or when additional constraints lead to a combinatorial explosion in the number of possible portfolios. Although cardinality is often imposed as a hard constraint in convex solvers, here it is treated as a soft constraint to allow for more flexible solutions.

In contrast to deterministic methods, evolutionary algorithms are particularly well equipped to handle such complexities. They excel in high-dimensional search spaces, can manage non-linear objectives effectively, and provide adaptive solutions as new data or constraints emerge. This project aims to address the non-trivial challenge of optimising portfolios with conflicting objectives, ultimately developing a robust framework that bridges theoretical models and practical investment strategies.

1.2 Relevance & Motivation

The motivation for this project stems from my long-standing interest in leveraging technology to solve complex financial problems. Building on my personal experience with trading bots and traditional back-testing frameworks, this project offers an opportunity to explore portfolio optimisation through a non-traditional lens. Rather than relying solely on standard single-

objective methods, such as maximising the Sharpe ratio or minimising Conditional Variance at Risk [2], it focuses on a multi-objective optimisation approach using evolutionary algorithms.

In addition to this, portfolio optimisation is set to become increasingly in demand in coming years as retail investing becomes more popular especially amongst younger people [3], and as such, this trend presents an opportunity to explore and develop innovative solutions within a rapidly growing market, driven by a new generation of investors whose preferences and risk profiles differ from previous cohorts.

The project is driven by both technical challenges and conceptual insights, aiming to bridge the gap between theoretical models and practical investment strategies. Ultimately, it seeks to demonstrate that a flexible, evolutionary computation framework can provide robust and realistic portfolio optimisation solutions in dynamic market environments.

1.3 Aims & Objectives

The overarching aim is to investigate whether Genetic Algorithms (GAs) when supplemented by other evolutionary techniques or not, can effectively address the multi-objective nature of portfolio optimisation. To achieve this, we explore three incremental setups increasing in complexity and real-world utility:

GA1 (Single-Objective Fitness Function, Single Portfolio Objective):

- A baseline GA that maximises a single performance metric (the Sharpe ratio). The aim of this GA is to validate the fundamental and foundational feasibility of using GAs for portfolio optimisation in the first place, and objectively compare its results to a standard deterministic MVO approach.

GA2 (Bi-Objective Fitness Function, Single Portfolio Objective):

- This secondary GA separates returns and risk into two objectives, with the aim of showcasing these pivotal trade-offs in portfolio optimisation so we can understand their significance. Here, a more advanced multi-objective GA approach is introduced providing a more nuanced understanding of efficient frontiers, as opposed to a single-objective function exploring one dimensional results as in GA1.

GA3 (Tri-Objective Fitness Function, Multi Portfolio Objectives):

- This final GA adds a third objective to represent real-world constraints (cardinality). It also incorporates a Particle Swarm Optimisation (PSO) inspired step to enhance convergence speed and efficiency of results. By allowing larger populations and more complex searches, this setup demonstrates how evolutionary algorithms can be adapted for practical and constrained financial decision making.

Chapter 2

Literature Review

This chapter reviews key reports and academic literature that have shaped the direction and methodology of this project.

2.1 Modern Portfolio Theory and Optimisation Challenges

Modern Portfolio Theory (MPT) was pioneered by Markowitz (1952) [1], who formulated portfolio selection as a mean-variance optimisation problem. In this framework, an efficient frontier of portfolios offers the best attainable trade-offs between expected return and risk (variance). The classical Markowitz model assumes that continuous asset weights can be solved as a convex quadratic program, yielding an optimal allocation for a given return target or risk tolerance. However, its practicality is limited by its simplifying assumptions. In real markets, financial analysts often impose additional constraints towards their investment decisions - for example, limiting the number of assets held in their portfolio (a cardinality constraint) or enforcing minimum/maximum holdings to each asset. Such constraints introduce combinatorial complexity, and adding a cardinality limit transforms the problem from a continuous quadratic program into a quadratic mixed-integer program (QMIP) [4], which is NP-hard. In other words, the inclusion of discrete investment decisions (whether to hold a particular asset) makes the search for an optimal portfolio computationally harder for large asset universes.

Traditional optimisation techniques (like branch-and-bound algorithms for mixed-integer problems) struggle to find optimal solutions in reasonable time as problem size and combinatorial complexity grows. This highlights a critical gap, especially with classical mean-variance methods that cannot easily handle real-world portfolio soft constraints like cardinality, render-

ing them less useful when such constraints are present. Researchers have long recognised these limitations of the mean-variance model and have sought alternative solution approaches.

2.1.1 Risk vs Return

In portfolio theory, risk and return inherently represent the two main conflicting objectives. Risk typically reflects the volatility or uncertainty of future returns (due to the stochastic nature of the stock market) and is often quantified by statistical measures such as variance or standard deviation. Return measures the expected profit from a portfolio or stock over a given period of time.

Balancing these two dimensions is central to the risk-return trade-off, which is graphically depicted by the efficient frontier; a curve representing the portfolios that offer the highest expected return for a given level of risk (and lowest risk for a given level of return). The efficient frontier provides a practical framework for investors to assess the trade-offs between taking on additional risk and the potential for higher returns, as underscored in Markowitz's work.

2.1.2 Mean Variance Optimisation

Mean-Variance Optimisation (MVO) is the classical formulation of Modern Portfolio Theory. It aims to either minimise portfolio variance (risk) for a target return value, or maximise expected return for a given risk tolerance. This is typically expressed as a quadratic programming problem that uses asset covariances to quantify portfolio risk. By adjusting asset weightings accordingly, MVO serves as a foundational approach in traditional portfolio construction to output a deterministic maximum Sharpe ratio and optimal portfolio weightings when given an input of stock data.

However, MVO's practicality is constrained when incorporating real-world constraints such as cardinality, which limits the number of assets in a portfolio. Traditional MVO models struggle with these constraints, as they transform the problem into a non-convex, NP-hard optimisation challenge, making it computationally intensive and less tractable [5]. This limitation opens avenues again for alternative approaches, such as Genetic Algorithms (GAs), which excel in exploring complex, non-convex search spaces. By employing evolutionary techniques, GAs can effectively navigate the expanded solution space introduced by cardinality constraints, offering more flexible and practical solutions in portfolio optimisation scenarios where MVO falls short.

2.1.3 Sharpe Ratio

The Sharpe Ratio extends beyond the basic risk-return comparison by incorporating risk-adjusted performance into the evaluation of portfolios. Instead of considering returns in isolation, the Sharpe Ratio divides the excess return (return above a risk-free rate) by the portfolio's standard deviation (risk). The risk-free rate is typically represented by the yield on government securities such as U.S. Treasury bills, which are considered free of default risk. The formula for the Sharpe Ratio is as follows:

$$\text{Sharpe Ratio} = \frac{R_p - R_f}{\sigma_p} \tag{2.1}$$

where:

- R_p = portfolio's expected return,
- R_f = risk-free rate,
- σ_p = standard deviation of the portfolio's returns (risk).

This ratio allows investors to compare portfolios on a more balanced basis, accounting for both reward and risk. A higher Sharpe Ratio indicates that the portfolio generates better returns per unit of risk, making it a pivotal evaluation metric for this project [6]. For this project, the risk-free rate will mostly be omitted from Sharpe Ratio calculations because in medium-term simulations with less data, or when comparing relative performance between portfolios, the risk-free rate tends to have minimal impact; especially when it is near zero or constant across portfolios [7].

2.1.4 Factor Models

To improve the accuracy of expected returns and manage exposure to hidden risk, factor models break down portfolio returns into explanatory components such as market factors, size, and value. A well-known industry-standard is the Barra Factor Model, which employs numerous factors to capture diverse sources of risk and potential return. However, Barra often requires proprietary data and subscriptions that are beyond the scope of many academic or individual projects including this one.

In this project, we will instead utilise the open-source Fama-French Factor Model [8] to derive more accurate expected returns in a later setup.

$$R_{it} - R_{ft} = \alpha_i + \beta_{iM}(R_{Mt} - R_{ft}) + \beta_{iSMB} \cdot SMB_t + \beta_{iHML} \cdot HML_t + \epsilon_{it} \tag{2.2}$$

The Fama-French approach, while less expansive than Barra, still accounts for key systematic risk factors (e.g., market, size and value) and aligns more closely with practical, real-world investment considerations than relying purely on historical arithmetic means from closing prices which could include anomalous fluctuations. The importance of these considerations was underscored during a conversation I had with the Head of EMEA Equities Technology at a tier-1 hedge fund, who emphasised that effective portfolio construction in industry increasingly hinges upon managing factor exposures. Controlling these exposures ensures portfolios remain aligned with strategic objectives while mitigating unintended risks, a crucial element in modern financial management.

2.2 Evolutionary Algorithms

Evolutionary algorithms (EAs) represent a class of heuristic optimisation techniques heavily inspired by Darwinian principles of biological evolution. Pioneered by researchers like Ingo Rechenberg and John Holland in the 1960s and 1970s [9], EAs encompass techniques like Genetic Algorithms (GAs), Ant Colony Optimisation [10] and Particle Swarm Optimisation (PSO) [11]. Each approach employs populations of candidate solutions that evolve over multiple iterations (also known as generations) via biological-like mechanisms such as reproduction, mutation and selection.

Unlike artificial intelligence methods such as deep learning for example, evolutionary algorithms do not rely on extensive data sets or explicit gradient information. Instead, they can navigate large search spaces by directly evaluating candidate solutions against an objective (e.g., a fitness function), making them notably suitable for problems with much more complex or ill-defined landscapes. This heuristic nature allows evolutionary algorithms to tackle highly diverse and uncharted tasks, such as popular optimisation and combinatorial problems.

Historically, evolutionary algorithms were more popular in the fields of optimisation and artificial intelligence, but their prominence had somewhat declined [12] with the rise of machine learning and the implementation of neural networks, which often provide more targeted and efficient approaches for solving specific problems with well-defined or narrow search spaces. Despite this, evolutionary algorithms and computational methods remain highly valuable, particularly for cases where:

- Exact models of the problem are unavailable or way too complex (e.g., problems in nature, design or strategy where rules are hard to express mathematically)
- Search spaces are very large, and the goal (objective function) is complex, non-linear or involves multiple conflicting objectives

A.E. Eiben and J.E. Smith's *Introduction to Evolutionary Computing* (2003) highlights the effectiveness of evolutionary algorithms in exploring large, multi-modal landscapes and handling noisy or dynamic environments [13]. While they emphasise the importance of parameter tuning and note the high computational cost due to iterative evaluations, the growth in modern processing power has reduced these barriers. As machine learning rose in prominence, particularly due to its success with large, labelled datasets and deep learning, evolutionary algorithms saw less mainstream attention. Nonetheless, they remain highly suitable for optimisation problems in domains requiring flexibility, adaptability, or where exact models are difficult to define.

2.3 Genetic Algorithms

Genetic Algorithms (GAs) are among the most widely adopted EAs, particularly suited for highly constrained problems. The typical process of a GA involves the following ordered setup:

1. **Initialisation**: randomly generate a population of candidate solutions
2. **Evaluation**: score each candidate's "fitness" (via a fitness function)
3. **Selection**: preferentially choose high-fitness solutions to breed the next generation
4. **Crossover**: combine "parent" solutions to form "offspring" that share features from both
5. **Mutation**: introduce small, random changes to some offspring to maintain diversity and avoid local optima
6. **Termination**: repeat steps 2-5 until a stopping condition is met (e.g., maximum generations)

To handle multi-objective problems, where solutions must balance competing goals, researchers often employ algorithms like Non-dominated Sorting Genetic Algorithm II (NSGA-II [14]). Rather than optimising a single fitness value, NSGA-II ranks solutions by non-dominance, where a solution is "dominated" if another solution is strictly better in at least one objective and equal or better in all others. This produces a Pareto front, or set of efficient trade-offs,

instead of a single ”best” solution.

In the context of our project, using NSGA-II allows for simultaneous optimisation of risk and returns (maximise returns while minimising risk) and potentially more objectives (e.g., cardinality).

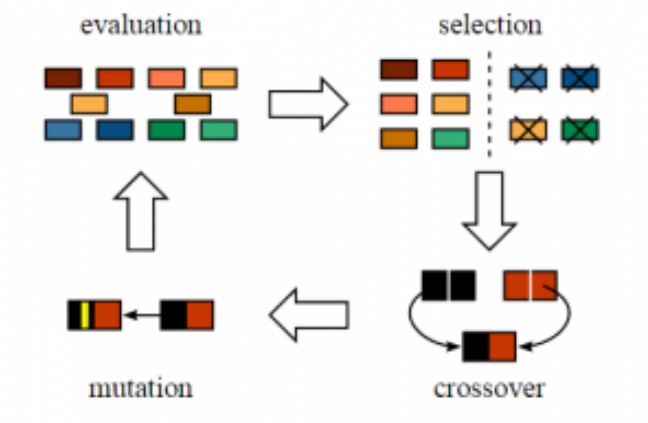


Figure 2.1: Core steps of a genetic algorithm [15]

Building upon the technical background of Genetic Algorithms (GAs) and Particle Swarm Optimisation (PSO), it's important to distinguish their roles. While GAs rely on biologically inspired operations like crossover and mutation to evolve solutions, PSO mimics the social behaviour of swarms. This is where particles adjust their positions in the solution space based on both personal experience and the success of their neighbours.

Unlike GA's stochastic exploration, PSO provides a more directed search toward local optima. Research such as the study ”Enhancing Genetic Algorithms using Multi Mutations” by Ahmad B. A. Hassanat [16] demonstrates how GAs can be refined using multiple mutation operators to enhance performance and avoid premature convergence. While Ahmad's work highlights the value of preserving diversity, it does not explore the integration of PSO-style refinement. In our project, we could address this gap by introducing a PSO-based local search step after the mutation phase, selectively applied to top-performing individuals. This targeted refinement could improve fitness by guiding elite candidates more effectively than random mutation alone, and as a result, this hybrid approach could reduce stochastic noise, maintain diversity, and accelerate convergence towards high-quality, multi-objective optimal solutions.

2.4 Optimisation Problems

While classical optimisation techniques such as gradient-based methods or branch-and-bound algorithms can solve simpler or convex variants of portfolio selection, finance problems in real life often defy these assumptions. A convex problem is one in which both the objective function and the feasible region form a convex set, meaning that for any two points within this set, the entire line segment connecting them also lies in the set.

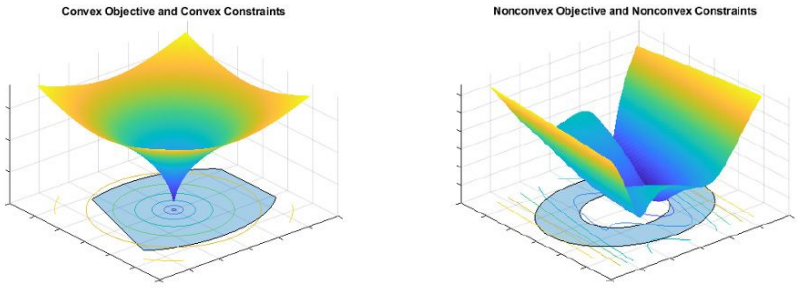


Figure 2.2: Convex vs Non-Convex Objectives in a 3D space [17]

In convex optimisation, any local minimum (a solution better than or equal to all feasible solutions in its vicinity) is also a global minimum (the best possible solution in the entire region). This property simplifies the search for optimal solutions because deterministic algorithms are guaranteed to converge to the global optimum under convexity.

On the other hand, non-convex problems which arise frequently in portfolio optimisation when real-world constraints are imposed can contain multiple local minima. Deterministic algorithms risk becoming trapped in these sub-optimal regions unless equipped with robust global search capabilities.

Moreover, portfolio optimisation problems are often non-deterministic, meaning that the objective function or constraints can be subject to a lot of uncertainty (e.g., fluctuating asset returns). Stochasticity plays a central role in addressing these challenges too. This refers to the elements of randomness in both the problem environment (e.g., unpredictable market conditions) and the optimisation process itself (e.g., random initialisation or mutation in evolutionary algorithms). As Eiben and Smith discuss [13] and as further demonstrated by Mitchell [18], evolutionary algorithms leverage this stochasticity as an advantage, incorporating random variations and probabilistic selection mechanisms. This approach allows EAs to explore vast, rugged search spaces more flexibly than deterministic methods, which often rely on precise gradient information and can be less robust when confronted with noise or incomplete information.

An equally important concept in optimisation problems is computational complexity, a measure of the time and memory required to solve a problem. Constraints such as minimum lot sizes, upper and lower bounds on asset allocations, and cardinality restrictions can transform what appears to be a straightforward quadratic programming task into a non-convex, NP-hard optimisation challenge. In these scenarios, the computational complexity can increase dramatically (often exponentially with the number of variables) and render classical techniques impractical for larger-scale problems. For instance, while Boyd and Vandenberghe [19] present a robust framework for convex optimisation, their methods are not directly applicable to these NP-hard settings. This limitation opens avenues for alternative approaches, such as Genetic Algorithms (GAs), which excel in exploring complex, non-convex search spaces. By employing evolutionary techniques, GAs can effectively navigate the expanded solution space introduced by cardinality constraints, offering more flexible and practical solutions in portfolio optimisation scenarios where MVO falls short.

2.5 Key Takeaways and Research Gaps

The literature review reveals several key insights and research gaps that inform the direction of this project:

1. **Limitations of Classical Portfolio Theory**: While Modern Portfolio Theory provides a solid foundation for portfolio optimisation, its practical applicability is severely limited when real-world constraints such as cardinality are introduced. The problem becomes non-convex and NP-hard, requiring alternative solution approaches.

2. **Strengths of Evolutionary Algorithms**: Evolutionary algorithms have demonstrated significant success in handling complex optimisation problems with multiple objectives and constraints. Their ability to navigate non-convex search spaces makes them particularly suitable for realistic portfolio construction scenarios.

3. **Complementary Nature of GA and PSO**: Genetic Algorithms excel at exploring discrete decision spaces (asset selection), while Particle Swarm Optimisation is efficient at refining continuous variables (asset weights). This complementary relationship suggests potential benefits from hybridisation.

4. **Need for Practical Constraints**: Real-world portfolio optimisation requires consider-

ation of practical constraints that are often overlooked in theoretical models. Cardinality constraints, transaction costs, and factor exposures significantly impact portfolio performance and should be incorporated into optimisation models.

Building on these insights, this project aims to address the identified gap by developing and evaluating a hybrid GA-PSO approach to portfolio optimisation. By fusing GA and PSO, we aim to jointly address the discrete and continuous facets of the portfolio optimisation problem. The GA mechanisms will drive exploration of which assets to include, efficiently navigating the combinatorial search of asset subsets, while PSO will guide the continuous optimisation of asset weightings for each candidate subset, quickly converging toward optimal risk-return allocations.

This hybrid approach is expected to handle return-risk multi-objectivity and cardinality constraints more effectively than traditional single-method approaches. Conceptually, the GA-PSO hybridisation can maintain genetic diversity to avoid PSO's premature convergence, while injecting PSO's learning capability to refine GA's outputs each generation. The end result would ideally be a more consistent and high-performing algorithm, capable of finding portfolios with better risk-return profiles under practical constraints.

Chapter 3

Requirements & Specification

To maintain clarity and realism, the requirements are divided into three categories: Functional Requirements, Non-Functional Requirements, and Optional/Nice-to-Have Requirements.

3.1 Functional Requirements

Our project must fulfil the following core objectives:

Table 3.1: Functional Requirements

Requirement ID	**Description**
FR1	Implement a genetic algorithm (GA) based portfolio optimisation system capable of processing a defined set of U.S. equities and evolving their allocations over multiple generations.
FR2	Integrate a robust fitness function that evaluates at least one industry-used portfolio optimisation goal (e.g., the Sharpe ratio function). [6]
FR3	Produce a Sharpe ratio for a given set of tickers that ideally is within 5–10% of the deterministic convex MVO solution [1] for the same set of tickers.
FR4	Provide visualisations to witness the GA's final Pareto front showing the trade-off between risk and returns, using Python libraries such as Matplotlib and/or Plotly.

Continued on next page

Table 3.1 – Continued

Requirement ID	**Description**
FR5	Implement a tri-objective genetic algorithm (GA3) that simultaneously optimises for return, risk, and cardinality, producing a comprehensive 3D Pareto front visualisation.
FR6	Integrate the Fama-French factor model in GA3 to generate expected returns based on systematic risk factors, replicating institutional-grade portfolio construction approaches.
FR7	Successfully implement PSO-inspired directional refinement post-mutation to improve convergence while maintaining genetic diversity across the solution space. [11]
FR8	Develop an adaptive mutation rate mechanism that dynamically adjusts based on population diversity metrics and convergence patterns.
FR9	Process and analyse at least 25 U.S. equities using stock data from the Alpha Vantage API spanning from January 2000 through December 2024.
FR10	Implement cardinality constraints allowing selection of up to 12 stocks from the universe, with appropriate penalty functions to enforce this constraint effectively.

3.2 Non-Functional Requirements

Our project is expected to have the following qualitative characteristics:

Table 3.2: Non-Functional Requirements

Requirement ID	**Description**
NF1	Outputs should be clearly presented both in the terminal and via interactive graphs using Python libraries such as Matplotlib or Plotly, displaying key metrics where possible (Sharpe ratio, expected return, risk, and cardinality) with appropriate colour schemes and legends.
NF2	Complete algorithm execution for 100 generations with a population size of 100 chromosomes should complete within 10 minutes on standard research hardware (i5 CPU, 16GB RAM), with performance scaling appropriately with increased population sizes.
NF3	The program must allow users to: select stocks by entering tickers, and adjust all key algorithm parameters (population size, generations, crossover/mutation rates)
NF4	Maintain a modular architecture that clearly separates concerns between algorithm, data processing, and visualisation components.
NF5	The system should incorporate at least two advanced evolutionary techniques (adaptive mutation, elitism, tournament selection, or particle swarm hybridisation) with improvement in solution quality over basic implementations.
NF6	Provide comprehensive error handling and user feedback, with meaningful error messages for API failures, data inconsistencies, and algorithm convergence issues.

3.3 Optional Requirements

These features would enrich our project and algorithm(s) but are not strictly required:

Table 3.3: Optional Requirements

Requirement ID	**Description**
OR1	Caching mechanism for stock data to accelerate repeated runs and improve development and testing efficiency, reducing API calls by at least 90% for subsequent executions.
OR2	Playback visualisation of the algorithm process (e.g., dynamic Pareto frontier updates) to offer immediate insight into convergence and stability across generations.
OR3	Integration of additional portfolio performance metrics beyond Sharpe ratio, such as Sortino ratio, maximum drawdown, and Value-at-Risk (VaR) calculations to provide comprehensive risk assessment.
OR4	Implementation of alternative evolutionary approaches (Differential Evolution, CMA-ES) as comparative benchmarks to evaluate the relative performance of our hybrid GA-PSO implementation.

3.4 Design Rationale

Genetic algorithms (GAs) were chosen as the core evolutionary algorithm for this project due to their robustness in handling discrete, multi-objective optimisation problems with complex constraints and search spaces as we encounter in portfolio optimisation. While all evolutionary algorithms (EAs) are known to excel in exploring non-deterministic spaces where no exact algorithm exists to solve the problem at hand, GAs are especially suited for scenarios involving multiple constraints and hybrid objective functions. Unlike Particle Swarm Optimisation (PSO), which performs best in smooth continuous landscapes or Ant Colony Optimisation (ACO), which is most ideal for routing and pathfinding on graphs, the distinct stages of GAs to use crossover, mutation and selection to evolve a population of candidate solutions makes them ideal for encoding possible portfolios as chromosomes with weights (genes), implementing constraints (like cardinality) and also multi-faceted objectives.

Their flexibility in handling multi-objective fitness functions, like in NSGA-II, and their extensibility to incorporate elements of other evolutionary algorithms such as PSO in post-mutation phases to solve the optimisation problem at hand (as we will do in setup 3) more efficiently, makes GAs the most appropriate choice for this context.

An important design choice for this project is also related to the data frequency used for portfolio construction. Monthly adjusted closing prices retrieved from the Alpha Vantage API are utilised instead of daily closing prices to minimise the high-frequency volatility that can distort risk and return estimates over longer horizons. This practice is commonly implemented in portfolio research [20], as daily or intraday data often introduce unnecessary noise, and even then, are much more expensive to acquire.

We specifically use data from January 2000 through December 2024, a range selected to capture multiple market cycles and maintain a balance between historical breadth and contemporary relevance. This period reflects a shift in market dynamics compared to prior decades, where the correlation between U.S. stock indices and inflation transitioned from negative in the 1980s and 1990s to strongly positive in the 2000s and 2010s [21], making post-2000 data potentially more indicative of current and future market behaviour, and therefore enhancing the robustness of our findings. While the exact start date is not too important to the validity of our project, as long as consistency is maintained, this choice mainly holds practical importance for real-world applicability and interpretability.

When it comes to the values we use in our objective functions, such as expected return, Setups 1 and 2 will utilise the simpler arithmetic mean of historical returns as a straightforward and interpretable estimate. Setup 3 however will incorporate the Fama-French factor model we touched upon earlier, to generate more accurate estimates of expected returns, closely mirroring institutional approaches that rely on factor exposures for portfolio construction. While some advanced parties like hedge funds employ proprietary models such as Barra, Fama-French offers a widely accessible alternative for academic and smaller-scale projects such as this one. By focusing on monthly data and factor-based returns, the project aims to produce optimisation outputs that are both realistically grounded and less prone to overfitting or short-term anomalies.

3.5 Formal Specifications

The project is structured around three main setups that progressively increase in complexity and real-world utility, as we explore each below.

3.5.1 GA1: Single-Objective Genetic Algorithm for Sharpe Ratio

This foundational setup implements a single-objective GA that maximises the Sharpe Ratio, one of the most established measures for evaluating risk-adjusted returns. It will serve as a baseline to verify whether our GA framework can approximate or come close to conventional Mean-Variance Optimisation (MVO) outcomes. The main rationale for this setup is as follows:

- **Sharpe Ratio function as a foundation:** Focusing on this well-known evaluation metric and single objective ensures straightforward benchmarking against standard results generated by MVO.
- **Proof of concept:** This setup will demonstrate whether portfolios encoded as chromosomes can be evolved via basic GA operations of selection, crossover, and mutation to produce viable stock allocations.

Specifications of GA1 are as follows:

- **Fitness function**: Maximise the Sharpe Ratio as a singular function.
- **Crossover**: Single-point crossover was for simplicity and baseline validation.
- **Mutation**: Simple mutation to maintain clarity and foundational understanding.
- **Selection**: Tournament selection is chosen for its balance between selective pressure and diversity.
- **Visualisation**: Graph showing highest Sharpe ratio plotted against generation to show convergence behaviour.

3.5.2 GA2: Bi-Objective Genetic Algorithm (NSGA-II) for Sharpe Ratio

Building on GA1, this configuration transitions to a bi-objective fitness function to simultaneously maximise returns and minimise risk using the NSGA-II framework. This setup illustrates how our evolutionary algorithm can explore trade-offs along a 2D Pareto front. The rationale for this genetic algorithm is as follows:

- **Multi-objective realism:** Better mimics actual portfolio decisions where multiple objectives must be considered, particularly risk vs. return trade-offs.
- **NSGA-II:** Enables non-dominated sorting to yield a Pareto front of efficient portfolios, allowing investor preference to guide final selection.

Specifications of GA2 are as follows:

- **Fitness function**: Bi-objective function to maximise return and minimise risk.
- **Crossover**: Simulated Binary Crossover (SBX) to generate more precise and realistic offspring.
- **Mutation**: Same as GA1.
- **Selection**: Tournament parent selection with NSGA-II selection for next generation.
- **Visualisation**: A 2D Pareto front with axes for risk and return.

NSGA-II is implemented in this setup as it is well-suited for this context due to its ability to handle conflicting objectives without requiring predefined weights, supporting investor-oriented solution diversity.

3.5.3 GA3: Tri-Objective Genetic Algorithm (NSGA-II) with Cardinality Constraints

GA3 extends GA2 by incorporating a third objective of cardinality to reflect real-world constraints such as transaction costs, capital limits, and portfolio rebalancing overheads [14]. Additionally, GA3 integrates expected returns calculated via the Fama-French Factor Model [8] and a PSO-inspired post-mutation refinement step. The rationale for this genetic algorithm is as follows:

- **Real-world constraints:** Introducing cardinality enables the model to reflect real portfolio size constraints, enhancing practical realism.
- **Fama-French Factor Model:** A more institutionally valid method to derive expected returns based on factor exposures.
- **PSO-Integration:** Introduces a directional search element to overcome the limitations of purely random mutation in traditional GAs.

Specifications of GA3 are as follows:

- **Fitness function**: Tri-objective to maximise return, minimise risk, and minimise cardinality.
- **Cardinality constraint**: Embedded as a penalty term to discourage overly complex portfolios.
- **Return estimation**: Derived using the Fama-French 3-Factor Model instead of historical averages.
- **Crossover**: Simulated Binary Crossover (SBX) as used in GA2.
- **Mutation**: Includes PSO-inspired refinement step to guide mutation directionally toward higher-quality regions.
- **Visualisation**: A 3D Pareto front with axes for risk, return, and cardinality.

The Fama-French Factor model is implemented in GA3 because this model better reflects institutional practices and improves return estimation by using explanatory factor data, and as such increases robustness and real-world applicability. Elements of Particle Swarm Optimisation (PSO) elements are also introduced in GA3, to make offspring adjusted towards the best-performing solutions in their neighbourhood post-mutation, allowing us to improve convergence rates while maintaining solution diversity.

Chapter 4

Methodology & Implementation

This section outlines the technological stack, data acquisition process, and the three main evolutionary setups used to address the different levels of complexity in our project.

4.1 Tooling & Technology

In this project, Python serves as the primary language, supplemented by libraries such as NumPy for numerical computations, Pandas for data manipulation, and Matplotlib or Plotly for data visualisation. All code is organised in VS Code via separate files:

- `GA1.py`, `GA2.py`, and `GA3.py` each implement a distinct genetic algorithm setup, building on one another.
- `data_fetch.py` handles data retrieval (from Alpha Vantage) and caching, reducing frequent API calls and staying within the constraints of the free Alpha Vantage tier.
- `FamaMonthly.txt` contains the open source factor data from Ken French's public research library.

This modular structure promotes code clarity and maintainability. By using Python's object-oriented capabilities, each genetic algorithm can be run independently for comparative experiments. Python was utilised because of its intuitive syntax and extensive ecosystem of scientific libraries heavily used in many research projects in industry. Among this, it is the language I am most proficient in, allowing for more efficient development. Alpha Vantage was chosen for retrieving stock data due to its extensive documentation and reliability, making it preferable over popular alternatives like yfinance for this project's needs. The codebase in general was

designed with simple scalability in mind, facilitating future extensions such as adding more complex GUI elements or integrating new objective functions.

The project codebase is structured as follows:

Listing 4.1: Project Directory Structure

```
project/
|- GA1.py                    # Single-objective GA implementation
|- GA2.py                    # Bi-objective GA implementation
|- GA3.py                    # Multi-objective GA with constraints
|- mvo.py                    # Deterministic MVO solver
|- data_fetch.py             # API handling and caching
|- cache_index.json          # Cached API responses
|- README.md                 # Project documentation
|- requirements.txt          # Python dependencies
|- .gitignore                # Git ignored files list
|- venv/                     # Python virtual environment
|- data/
|  - FamaMonthly.txt       # Fama-French monthly factors
|  - stocks.csv            # Historical price data (1)
|  - stocks2.csv           # Historical price data (2)
|- tests/
   - test_data_fetch.py    # Tests for data_fetch.py
   - test_ga1.py           # Tests for GA1
   - test_ga2.py           # Tests for GA2
   - test_ga3.py           # Tests for GA3
   - test_mvo.py           # Tests for mvo.py
```

The main libraries and frameworks used in the setups are as follows:

1. **NumPy**

   Utilised for its fast, vectorised operations and comprehensive mathematical functions, making it ideal for performing calculations in our program. It is also the most commonly used library to perform numerical operations, and as such, presents itself as a perfect choice. [22]

2. **Pandas**

   Chosen for its powerful data manipulation capabilities, particularly in handling time-series financial data and simplifying data preparation tasks. [23]

3. **DEAP (Distributed Evolutionary Algorithms in Python)**

   Selected because it provides a robust and well-documented framework for exploring evolutionary computation in Python, including genetic algorithms. It is widely adopted in research settings. [24]

4. **Plotly**

   Used for its interactive plotting features that enable dynamic and more in-depth visualisation of plotted points among Pareto fronts and convergence, enhancing user insight. [25]

5. **Argparse**

   Preferred for its simplicity in parsing command-line arguments, making the program highly configurable without altering the codebase. [26]

6. **Rich**

   Adopted for its ability to render aesthetically pleasing and informative console outputs (with tables and panels, for example), improving readability and presentation of findings. [27]

7. **Built-in Libraries: `random`, `warnings`, `time`, `sys`**

   Leveraged for their essential functions like random number generation and system operations to ensure a lightweight and reliable codebase.

Additional Details:

- **Development Environment:** All development was done using VS Code in a Python 3.11 environment on macOS. Key packages were managed using `pip` and virtual environments.
- **Version Control:** Git was used throughout development for version control to maintain stability.

4.2 Data Acquisition

The Alpha Vantage API is employed to retrieve monthly adjusted closing prices for U.S. equities from January 2000 to December 2024. Due to API constraints (25 requests/day), a local caching mechanism is implemented in `data_fetch.py`, which saves retrieved price data to a `stocks.csv` file but caches stock data locations in a file `cache_index.json`. This approach prevents redundant API calls and speeds up repeated experimental runs.

The program follows a modular structure where the primary interface is the `get_stock_data()` function. This function accepts a list of tickers and determines whether to fetch fresh data or load from a previously saved cache. Tickers are first normalised and sorted alphabetically to form a consistent cache key. If cached data is available (tracked using a `cache_index.json` file), it is loaded directly using `pandas.read_csv()`; otherwise, the `fetch_data()` function is called, which uses the `alpha_vantage` Python package to retrieve and clean monthly adjusted close prices. The results are saved to a timestamped CSV file within a local `data/` directory, and the cache index is updated accordingly via `save_cache_index()`. Command-line interaction is handled through the `main()` function, which uses the `argparse` library to allow users to specify ticker symbols and optionally force a refresh from the API.

Alongside the market data, the project also utilises a `FamaMonthly.txt` file sourced from Ken French's website, containing Fama-French factor data for the same date range. This factor data is particularly relevant in Setup 3, where factor-based expected returns offer a more realistic metric than pure historical averages.

By combining these two data sources, the project is able to simulate market conditions and evaluate portfolio performance under different scenarios, including constraints on the number of assets (cardinality) and additional factor-based considerations. This setup enables reproducible experimentation while staying within practical data access limits.

4.2.1 High Level Architecture

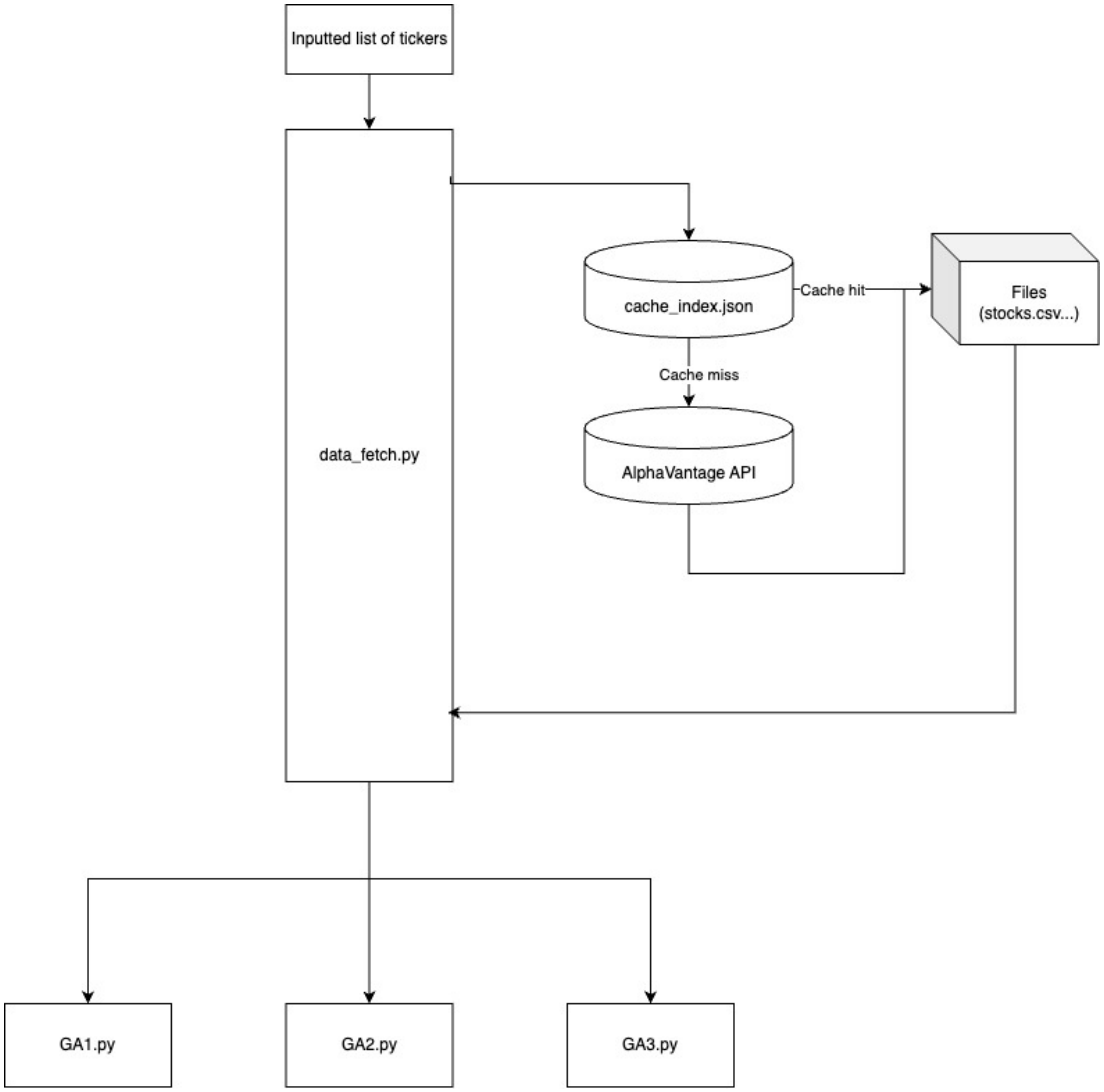


Figure 4.1: Data Acquisition UML Diagram

4.3 Genetic Algorithm Implementation

This section details the technical implementation of the key genetic algorithm components across our three setups (GA1, GA2 and GA3), highlighting code structures, implementation choices, and progressive enhancements through each setup. First, we will talk about the universal implementations across all three setups, then separately define the specifics for each setup in its own domain.

4.3.1 Universal Implementations

Data Handling

All three GAs share a common data processing pipeline implemented by the `prepare_data()` function. This function loads the (monthly adjusted) stock price data from the CSV files, computes monthly returns and prepares the necessary data structures to be interpreted by our genetic algorithm. This modular approach allows us to separate data preparation from the genetic algorithm logic and make the code more maintainable and easier to test. The function returns a dictionary containing all the relevant data which is then unpacked in the main function later in the program.

Parameter Configuration

The foundational parameters for all of our genetic algorithms are configured using a combination of default values and command-line arguments to facilitate experimentation for the user. We utilise the `argparse` library to enable the dynamic configuration of our key GA parameters and program itself through command-line arguments. This facilitates rapid experimentation without having to hardcode parameters and options into the file.

Listing 4.2: Command Line Argument Parser for NSGA-II

```
def parse_arguments():
    """Parse and return command line arguments."""
    parser = argparse.ArgumentParser(description="NSGA-II Portfolio
        Optimisation")
    parser.add_argument("-g", "--graph", action="store_true",
                        help="Display Pareto front graph")
    parser.add_argument("-p", "--pop_size", type=int, default=100,
                        help="Population size for the GA")
    parser.add_argument("-n", "--ngen", type=int, default=50,
```

```
                        help="Number of generations")
    parser.add_argument("-m", "--mut_rate", type=float, default=0.1,
                        help="Mutation rate for the GA")
    parser.add_argument("-a", "--animate", action="store_true",
                        help="Create animated visualisation of Pareto
                            front evolution")
    parser.add_argument("-r", "--runs", type=int, default=1,
                        help="Number of NSGA-II runs for sensitivity
                            testing (if > 1, performs multiple runs)")
    return parser.parse_args()
```

The **graph (–graph)** parameter if passed, allows us to witness the graph of the specific setup. For setup 1 (GA1.py) this is simply the Sharpe ratio against the generation number. For setup 2 (GA2.py) this is the returns vs risk value generating a pareto front graph. And for setup 3 (GA3.py) this is the returns vs risk vs cardinality 3D plot. All graphs automatically open up in the browser of the device on a localhost port via the Plotly library, and are interactive allowing the user to hover over points to view portfolio allocations on that specific point. The choice of Plotly over Matplotlib here is due to this specific reason of being able to interact with the graph more so, especially on a 3 dimensional scale.

The **population size (–pop_size)** parameter, with a default of 100 as specified in our design section, alters the population size parameter for the whole GA.

The **mutation rate (–mut_rate)** parameter has a default value of 0.1, and is passed to the mutation function in the corresponding setup. For setup 1 and 2, this is passed to the `simple_mutation()` function as the **indpb** (short for individual probability) parameter.
The **number of generations (–ngen)** parameter, with a default of 100 also specified in our design section, sets the number of evolutionary generations our algorithm goes through and is essentially the stopping condition of the program in most cases, especially for GA1 and GA2.

Console Logging

A highly relevant technical implementation detail across all setups is our use of the **Rich** library to create a sophisticated console-based user interface. Instead of relying on simple print statements with no colour or organisation, we implement a structured and visually appealing output system that enhances usability and information presentation.

The Rich library allows for coloured text, tables, panels, and other formatting elements that make the output both more readable and informative. We specifically implement:

- Formatted headers with dataset information (time period, number of assets)
- Tabular presentation of GA parameters for clear configuration visibility
- Structured performance summary tables with proper alignment and formatting
- Colourised output to highlight important metrics like the best Sharpe ratio

This implementation choice significantly improves the user experience by organising large amounts of numerical data in a digestible format.

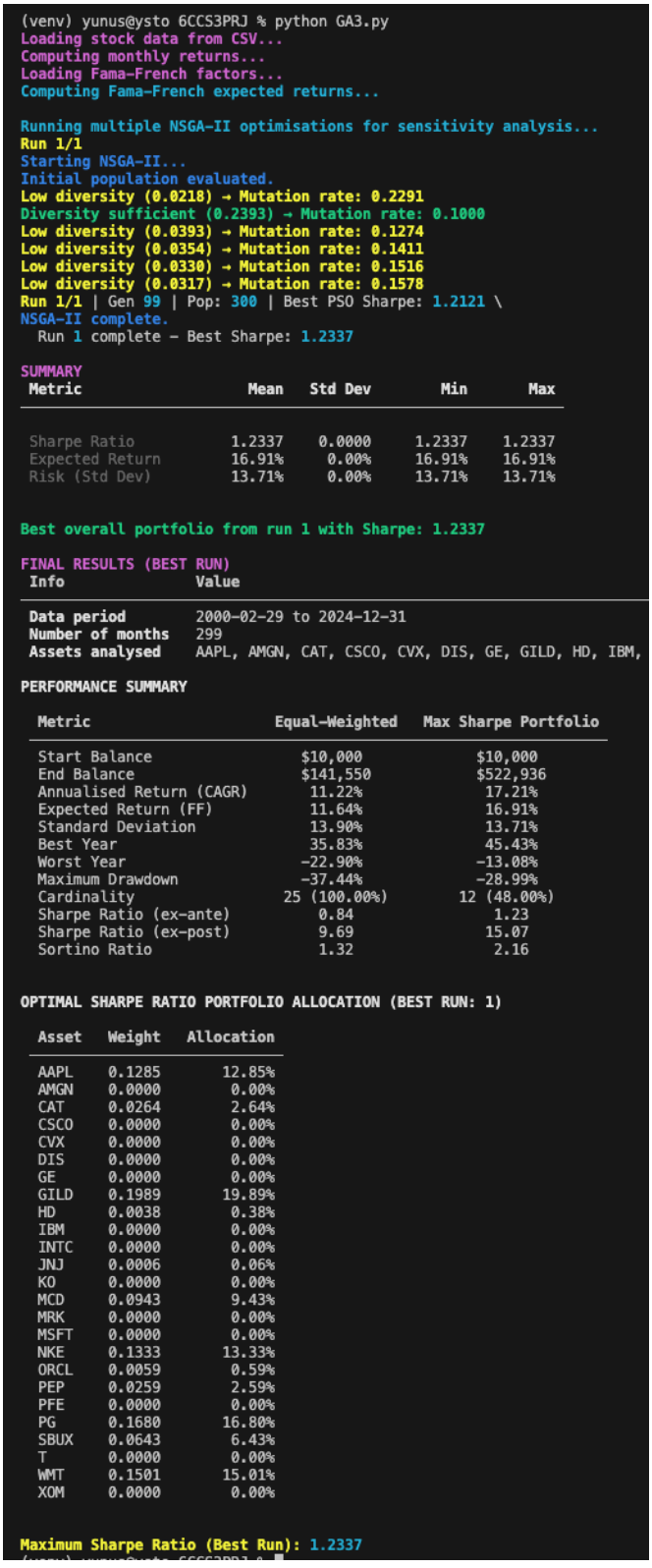


Figure 4.2: Console with Rich formatting

Visualisation Framework

All of the GAs utilise the Plotly library for data visualisation to generate and show the interactive graphs. Plotly has been implemented instead of Matplotlib due to its more interactive capabilities, which provide a more insightful exploration of the results and improve the utility for end users. The interactivity is particularly valuable for portfolio optimisation as it allows users to:

- Hover over points to view detailed portfolio allocations
- Zoom into specific regions of the Pareto front more smoothly on a browser
- Rotate and explore points on 3D visualisations more clearly

Each setup implements progressively more complex visualisations as follows:

GA1: Shows the plot of Sharpe Ratio against Generation number, providing insight into convergence behaviour and rate.

GA2: Shows the plot of Risk vs Expected Return (Pareto frontier), enabling visualisation of the risk-return trade-off.

GA3: Shows the plot of Risk vs Expected Return vs Cardinality (3D Pareto frontier), illustrating the complex trade-offs between three competing objectives.

The visualisation implementation includes careful attention to data formatting and informative labelling, and as such this enhances the research utility of our implementation by making complex portfolio optimisation results more accessible and interpretable.

Population and Chromosome Implementation

The population and chromosome representations are implemented consistently across all three setups using DEAP framework's toolbox pattern. Our implementation follows these key steps:

1. First, we initialise the DEAP genetic algorithm framework by creating custom fitness and individual classes. For the chromosome representation, we use real-valued floating-point arrays where each element represents the weight allocated to a specific asset in the portfolio. This representation directly maps to allocation percentages, making the results immediately interpretable in a financial context.

2. Then, population initialisation is handled through the toolbox's registration mechanism. The `attr_weight()` function generates random values between 0 and 1 for each gene (asset

weight), while the individual and population functions create structured instances using these random weights. This approach creates a diverse initial population of portfolios with random asset allocations.

3. Following this, another critical aspect of our chromosome implementation is the repair function, which ensures all portfolio weights sum to 1.0 (representing 100% allocation of capital). This function is applied after every genetic operation to maintain this constraint, to handle edge cases such as zero-sum weights by defaulting to equal allocation across all assets in the worst case. The repair mechanism exemplifies how we integrate domain-specific constraints directly into the genetic algorithm framework.

Selection Implementation

The chosen method of parent selection being tournament selection, is implemented consistently across all three setups as our primary selection mechanism for choosing parent portfolios for reproduction. We selected tournament selection specifically for its balanced preservation of genetic diversity while maintaining appropriate selection pressure.

Our implementation uses DEAP's built-in **selTournament** function with a tournament size parameter that varies slightly between setups. For GA1, we use a tournament size of 2, which provides moderate selection pressure appropriate for single-objective optimisation. However, in setups 2 and 3 we increase the tournament size to 3 to provide stronger selection pressure for the more complex multi-objective problems.

Tournament selection works by randomly sampling small subsets of the population (of size equal to the tournament size) and selecting the best individual from each sample to become a parent. This method has several advantages for our portfolio optimisation task:

1. It handles negative fitness values effectively, which is crucial when evaluating portfolio performance during market downturns

2. It maintains reasonable diversity in the population by giving lower-ranked solutions a chance to be selected

3. It offers computationally efficient performance compared to alternatives like roulette wheel selection

4. It naturally scales to multi-objective optimisation scenarios in GA2 and GA3

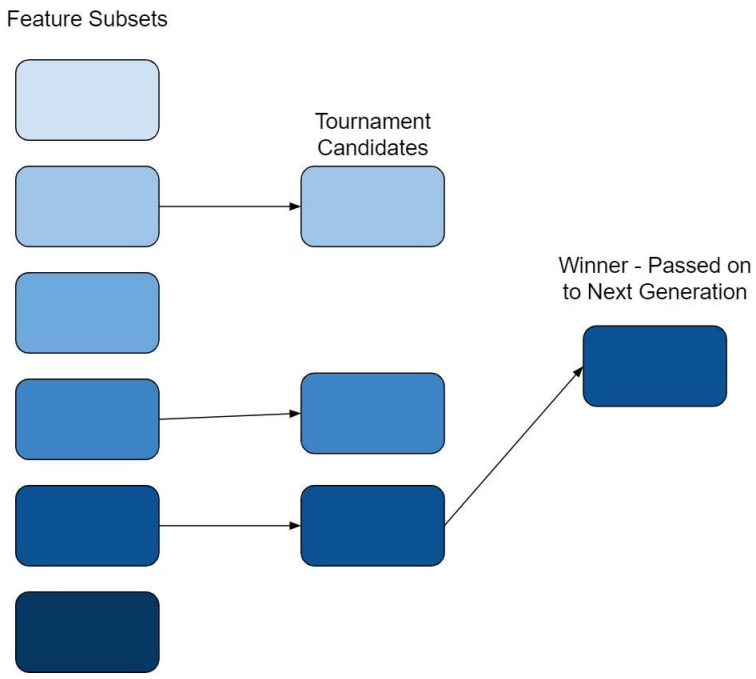


Figure 4.3: Tournament selection diagram [28]

While tournament selection remains consistent across setups for parent selection, the criteria for determining the ”best” individual evolves from a single Sharpe ratio in Setup 1 to non-dominated ranking and crowding distance in Setups 2 and 3. This progression demonstrates how a consistent selection mechanism can adaptively handle increasingly complex optimisation scenarios.

4.3.2 GA1 Implementation

This setup serves as our foundational implementation, focusing on maximising the Sharpe ratio through a single-objective genetic algorithm. While building on the universal components discussed earlier, this setup has several unique implementation aspects that establish our baseline approach to portfolio optimisation.

Single-Objective Fitness Function Implementation

The heart of our first setup (GA1) is the **evalPortfolio()** function, which implements our single-objective fitness evaluation focused solely on the Sharpe ratio, as you may recall from the earlier background chapter.

Listing 4.3: GA1. Fitness Function

```
def evalPortfolio(individual):
    weights = np.array(repair(individual))
    exp_ann_return, ann_vol = portfolio_performance(weights,
        monthly_returns)
    sharpe = compute_sharpe(exp_ann_return, ann_vol)
    return (sharpe,)

toolbox.register("evaluate", evalPortfolio)
toolbox.register("mate", tools.cxUniform, indpb=0.5)
```

This function integrates directly with the DEAP framework through registration with the toolbox. The DEAP toolbox is a modular framework that registers and manages the core components of our genetic algorithm, including how individuals are initialised, evaluated, mutated, and selected. In this setup, it links our custom **evalPortfolio()** function to evaluate each portfolio based on the Sharpe ratio function, registers uniform crossover and mutation operations, and uses tournament selection to choose parents. This integration allows DEAP to handle the evolutionary process efficiently, guiding the population toward higher-performing portfolios across generations.

Our fitness function in this setup specifically takes each candidate portfolio (chromosome), normalises its weights through the repair function, calculates the expected return and volatility, and then computes the Sharpe ratio. The fitness evaluation process also relies on the `portfolio_performance()` and `compute_sharpe()` helper functions, which handle the financial calculations using vectorised NumPy operations for efficiency.

The function is registered with the DEAP toolbox, making it the sole criterion for evaluating portfolio quality in this setup. This focused approach allows us to directly compare the GA's ability to optimise the Sharpe ratio against traditional approaches like Mean-Variance Optimisation.

Crossover Implementation

In GA1, we implement simple uniform crossover using DEAP's built-in **cxUniform** operator. This crossover method works by exchanging individual genes (asset weights) between two parent portfolios with a specified probability (0.5 in our implementation stated from our design section). The crossover probability (**cxpb**) parameter is set to 0.9, meaning that 90% of selected pairs will undergo crossover.

After crossover, we apply the repair function to both offspring to ensure weight normalisation. This implementation choice balances the exploration of new weight combinations while maintaining the integrity of successful partial allocations. The uniform crossover is particularly well-suited for our portfolio optimisation task as it allows fine-grained mixing of allocation strategies from two parent portfolios.

Listing 4.4: Example of Simple Uniform Crossover in GA1

```
# Parent A and Parent B (asset weight vectors)
parent_a = [0.2, 0.1, 0.3, 0.4]
parent_b = [0.1, 0.3, 0.2, 0.4]

# Apply uniform crossover with indpb=0.5
child_a, child_b = tools.cxUniform(parent_a[:], parent_b[:], indpb=0.5)

# Example result (randomly chosen gene swaps)
# child_a = [0.1, 0.1, 0.3, 0.4]
# child_b = [0.2, 0.3, 0.2, 0.4]

# Apply repair to ensure valid allocations
child_a = repair(child_a)
child_b = repair(child_b)
```

Mutation Implementation

Mutation in Setup 1 is handled by our custom `simple_mutation()` function, which iterates through each gene (asset weight) and, with a probability equal to the mutation rate, replaces it with a new random value between 0 and 1. This approach creates more significant changes to our portfolio weights compared to incremental adjustments, allowing for more aggressive exploration of the solution space.

Like with crossover, mutation is followed by applying the repair function to normalise weights. The mutation rate (**mutpb**) parameter is set to 0.1 by default, meaning approximately 10% of individuals in the population will undergo mutation in each generation. This implementation strikes a balance between maintaining successful portfolios and exploring new allocation strategies.

The implementation of setup 1 establishes a solid foundation for portfolio optimisation using genetic algorithms, demonstrating that even a simple single-objective GA can effectively find portfolios with risk-adjusted returns. The structure is designed for extensibility, setting the stage for the more complex multi-objective implementations in subsequent setups.

4.3.3 GA2 Implementation

This setup builds upon our foundational implementation from setup 1 by transitioning to a multi-objective optimisation approach using the NSGA-II algorithm. It maintains the core genetic algorithm components from setup 1 but introduces more significant enhancements to address the dual objectives of maximising returns while minimising risk.

Bi-Objective Fitness Function Implementation

The central enhancement in GA2 is the implementation of a bi-objective fitness function through our **evalPortfolio()** function. Unlike GA1's single-objective focus on the Sharpe ratio, this implementation explicitly separates the return and risk components:

Listing 4.5: GA2 Fitness Function

```
def evalPortfolio(individual):
    weights = np.array(repair(individual))
    exp_ann_return = calc_expected_return(weights, mean_monthly, freq)
    ann_vol = np.sqrt(np.dot(weights.T, np.dot(cov_monthly * freq,
        weights)))

    # Bi-objective: minimise negative expected return and risk
    return -exp_ann_return, ann_vol

toolbox.register("evaluate", evalPortfolio)
```

The fitness function now returns a tuple containing the negative of the expected return (for minimisation) and the portfolio volatility. This implementation aligns with DEAP's convention for multi-objective optimisation, where each objective's weight in the fitness function indicates whether it should be maximised (positive weight) or minimised (negative weight).

To support this multi-objective approach, we modified the DEAP setup to create a **FitnessMulti** class with weights (-1.0, -1.0), indicating that both objectives should be minimised. And since we want to maximise returns, we negate the expected return value, and so this implementation allows the NSGA-II algorithm to construct a Pareto front of non-dominated solutions, representing the trade-off between risk and return.

NSGA-II Selection Implementation

A key implementation difference in GA2 compared to GA1 is the replacement of the simple tournament selection and elitist approach from GA1 with NSGA-II's specialised selection method. This is implemented through the **tools.selNSGA2** function, which selects individuals based on Pareto dominance ranking and crowding distance.

This implementation ensures diversity along the Pareto front by preserving solutions that represent different trade-offs between risk and return. The NSGA-II selection mechanism is particularly effective for portfolio optimisation as it naturally maintains a diverse set of investment strategies, from conservative (low risk, low return) to aggressive (high risk, high return).

Simulated Binary Crossover Implementation

GA2 also introduces Simulated Binary Crossover (SBX) through DEAP's **cxSimulatedBinaryBounded** function, replacing and improving upon the uniform crossover from GA1.

This implementation uses an eta parameter of 15, which controls how similar offspring are to their parents (higher values produce offspring more similar to parents). The low and up parameters enforce bounds on the gene values, ensuring they remain between 0 and 1 before the repair function normalises them.

SBX is particularly well-suited for real-valued optimisation problems like portfolio allocation as it better preserves the distance properties between parent solutions in the offspring. This implementation choice helps maintain the structured exploration of the solution space, which is crucial for identifying the Pareto front effectively.

Simulated Binary Crossover (SBX) Example

Simulated Binary Crossover (SBX) is a recombination operator used in real-valued genetic algorithms to generate offspring that truly inherit properties from their parents while preserving the distance structure of the solution space.

Unlike a simple uniform crossover which just randomly swaps gene values; SBX lets us produce offspring that are truly genetically related to their parents controlled by a distribution index (η). For two parent genes, x_1 and x_2, SBX works as follows:

1. First, a random number u is drawn from a uniform distribution $U(0, 1)$.
2. Then, the spread factor β_q [29] is computed using:

$$\beta_q = \begin{cases} (2u)^{\frac{1}{\eta+1}}, & \text{if } u \leq 0.5, \\ \left(\frac{1}{2(1-u)}\right)^{\frac{1}{\eta+1}}, & \text{if } u > 0.5. \end{cases}$$

3. The offspring genes x_1' and x_2' are then generated by the formulae:

$$x_1' = 0.5\left[(1+\beta_q)x_1 + (1-\beta_q)x_2\right],$$

$$x_2' = 0.5\left[(1-\beta_q)x_1 + (1+\beta_q)x_2\right].$$

Example:

Assume we have two parent genes:

$$x_1 = 0.2 \quad \text{and} \quad x_2 = 0.8,$$

which have a distribution index of $\eta = 15$. Let the randomly generated number be $u = 0.3$. Since $u \leq 0.5$, we use the first branch:

$$\beta_q = (2 \times 0.3)^{\frac{1}{15+1}} = (0.6)^{\frac{1}{16}} \approx 0.9686.$$

Now we can compute the offspring:

$$x_1' = 0.5\left[(1+0.9686)\cdot 0.2 + (1-0.9686)\cdot 0.8\right],$$

$$x_1' \approx 0.5\left[1.9686 \times 0.2 + 0.0314 \times 0.8\right],$$

$$x_1' \approx 0.5\,[0.3937 + 0.0251] \approx 0.5 \times 0.4188 \approx 0.2094.$$

And similarly for the other formula,

$$x_2' = 0.5\,[(1 - 0.9686) \cdot 0.2 + (1 + 0.9686) \cdot 0.8]\,,$$

$$x_2' \approx 0.5\,[0.0314 \times 0.2 + 1.9686 \times 0.8]\,,$$

$$x_2' \approx 0.5\,[0.0063 + 1.5749] \approx 0.5 \times 1.5812 \approx 0.7906.$$

Therefore, the offspring are approximately:

$$x_1' \approx 0.2094 \quad \text{and} \quad x_2' \approx 0.7906.$$

This example demonstrates that SBX creates offspring that remain genetically closer to the parents than simple crossover, preserving the relative distances between solutions. This controlled recombination is particularly beneficial in optimisation problems such as portfolio allocation, where maintaining the structure of candidate solutions helps in efficiently exploring the solution space and converging to the Pareto front.

In our implementation (see `GA2.py`), portfolios are encoded as individuals with genes representing asset allocation weights. By employing DEAP's **cxSimulatedBinaryBounded** function, we apply SBX directly to these weights, ensuring that:

- Offspring portfolios inherit a more similar asset allocation structure as their parent than before
- Asset weights remain within the valid bounds (0 to 1)
- The distribution index of $\eta = 15$ directly controls the similarity between parents and offspring, preserving the diversification and risk-return properties of the original portfolios.

This strategy is integral to our multi-objective optimisation framework. By maintaining the core characteristics of high-performing portfolios, our approach facilitates a structured exploration of the solution space and efficient convergence toward a Pareto front that optimally balances expected return and risk.

Max Sharpe Portfolio Implementation

While GA2 generates a Pareto front of solutions, we should still select a single ”best” portfolio for comparison with the equal-weighted benchmark. We implement this by finding the portfolio with the highest Sharpe ratio among all solutions on the Pareto front:

Listing 4.6: Finding best portfolio from Pareto Front

```
def find_best_portfolio(pareto_front, monthly_returns):
    """Find the portfolio with the highest Sharpe ratio on the Pareto
        front."""
    best_portfolio = None
    best_sharpe = -np.inf
    for ind in pareto_front:
        weights = np.array(repair(ind))
        exp_return, risk = portfolio_performance(weights,
            monthly_returns)
        sharpe = compute_sharpe(exp_return, risk)
        if sharpe > best_sharpe:
            best_sharpe = sharpe
            best_portfolio = weights
    return best_portfolio, best_sharpe
```

This implementation demonstrates how the multi-objective approach can still be reconciled with traditional portfolio selection criteria. By generating the entire efficient frontier and then selecting based on the Sharpe ratio, we combine the advantages of both approaches.

GA2 as a whole represents a significant advancement over GA1 by explicitly modelling the risk-return trade-off that is central to portfolio theory. The NSGA-II algorithm effectively identifies the efficient frontier of portfolios, providing investors with a comprehensive view of their options while still allowing for the selection of a specific portfolio based on risk preferences or other criteria. This implementation serves as a bridge between the simple single-objective approach and the more complex three-objective model in GA3.

4.3.4 GA3 Implementation

This final setup is the magnum opus of our project and extends our portfolio optimisation framework to a three-objective problem, incorporating a cardinality constraint in addition to risk and return. By building upon the multi-objective foundation of GA2, Setup 3 demonstrates how to handle additional real-world constraints and more nuanced fitness evaluations while still utilising the same DEAP-based infrastructure.

Fama-French Factor-Based Expected Returns

A key enhancement in Setup 3 is the use of Fama-French factors to compute asset-specific expected returns, replacing the simpler mean returns from the previous setups. These factors (Market, SMB, HML, and the risk-free rate) allow for a more fine-grained estimation of expected returns based on established financial models:

Listing 4.7: FamaMonthly.txt data sample starting from January 2000

```
        Mkt-RF      SMB      HML       RF
200001,    -4.74,     5.77,    -1.88,     0.41
200002,     2.45,    21.36,    -9.59,     0.43
200003,     5.20,   -17.20,     8.13,     0.47
200004,    -6.40,    -6.68,     7.26,     0.46
200005,    -4.42,    -6.05,     4.75,     0.50
...
```

1. Data Merging
    - Monthly returns for each asset and the Fama-French three-factor data are loaded.
    - Both datasets are aligned by their time indices so that excess returns (i.e., asset returns above the risk-free rate) can be accurately computed.
2. Regression for Alpha and Betas
    - For each asset, an ordinary least squares (OLS) regression is performed on its excess returns against the three Fama-French factors.
    - From these regressions, the intercept (*alpha*) and factor loadings (*betas*) are extracted. These are then combined with the average factor values to estimate the expected monthly return of each asset.
3. Final Expected Returns

- The resulting monthly expected returns are aggregated and passed into the `evalPortfolio` function to compute annualised returns as part of the GA's objective function.
- This approach aligns more closely with financial theory and provides a robust, market-informed basis for return expectations.

Triple-Objective Fitness Function Implementation

In this fitness function now we build on top of the one we saw in GA2, with a third objective of cardinality in addition to risk and return:

1. Returns (maximise)
2. Risk (minimise)
3. Cardinality (minimise)

Using DEAP, specify these three objectives with a **weights=(-1.0, -1.0, -1.0)** to ensure the algorithm simultaneously minimises our negative return, risk and number of assets used.

4.3.5 Advanced GA Operators Implementation

While the NSGA-II selection mechanism for multi-objective optimisation again remains similar to GA2, we incorporate additional advanced genetic operators to enhance the search for optimal portfolios:

1. Adaptive Mutation Rate

We introduce a dynamic mutation mechanism to help avoid premature convergence in the population. After performing crossover, the `adaptive_mutation_rate()` function is called to determine the mutation probability (`mutpb`) for the upcoming mutation step. The process involves:

1. **Population Diversity Computation**

   The function `compute_population_diversity(offspring)` is used to calculate the standard deviation of each gene (asset weight) across all individuals in the population. These are averaged to produce an overall diversity score.

2. **Threshold-Based Adjustment**

- If the diversity score falls below a defined threshold (e.g., 0.05), the mutation rate is increased proportionally, up to a capped maximum (e.g., 0.5).
- If the diversity remains above the threshold, the base mutation rate (e.g., 0.1) is retained.

3. **Implementation in Code**

Listing 4.8: GA3 Adaptive Mutation

```
# -- adaptive mutation
current_mutpb = adaptive_mutation_rate(offspring,
                    base_mutpb=base_mutpb,
                    diversity_threshold=0.05,
                    max_mutpb=0.5)
# -- mutation
for mutant in offspring:
    if random.random() < current_mutpb:
        toolbox.mutate(mutant)
        mutant[:] = repair(mutant)
        del mutant.fitness.values
```

This approach injects additional randomness at critical points, helping the algorithm escape local optima by encouraging exploration of new regions of the search space.

2. PSO Local Search

After crossover and mutation (but before final selection), we further refine the most promising offspring through a brief Particle Swarm Optimisation (PSO) step. This local search improves individuals by iteratively optimising their Sharpe ratio:

1. **Identifying Top Offspring**

   We compute the Sharpe ratio for each individual and select the top 10% (or at least one individual) based on their performance.

2. **PSO Procedure**

   - *Initialisation:* Each selected individual is cloned into a small swarm (e.g., 10 particles), each with slight random perturbations.
   - *Update Equations:* Over a few iterations (e.g., 5), particles update their velocity and position based on standard PSO update rules.

- *Repair & Evaluation:* After each update, weights are normalised to sum to 1, and Sharpe ratios are recalculated.
- *Result:* The particle with the highest Sharpe ratio replaces the original offspring.

3. **Implementation in Code**

   The `pso_local_search()` function spawns particles (`num_particles=10`) and runs for a few iterations (`num_iterations=5`), returning the best-performing weight vector.

By combining genetic algorithms with targeted local search, we achieve a hybrid approach that balances broad exploration (via GA) with focused exploitation (via PSO), especially effective in multi-objective settings.

4.4 Testing

In our project, we embraced a comprehensive testing approach using Python's built-in unittest library to ensure the reliability and correctness of our codebase. The unittest framework provided a structured way to verify that each component of our financial system functioned as expected, all the way from initially data fetching to executing the genetic algorithm implementations. By writing systematic test cases for each module, we were able to isolate and validate individual components, ensuring they met specifications before integration into the larger system.

Our testing strategy focused on creating a robust suite of tests that verified not only the basic functionality; but also edge cases and error handling mechanisms. As shown in Table 4.1, we achieved an overall test coverage of 95%, with most files exceeding 96% coverage. This high level of test coverage is crucial when developing financial applications [30] where reliability is paramount. In a practical implementation, portfolio optimisation algorithms may guide investment decisions involving millions in capital; therefore, thoroughly tested code is not only good practice but an essential requirement. The only module with relatively lower coverage (80%) was GA3.py, which contains our most advanced optimisation techniques. Future work could prioritise improving coverage for this module, particularly around the more complex multi-objective optimisation functions.

Name	Stmts	Miss	Cover (%)
GA1.py	196	7	96
GA2.py	374	11	97
GA3.py	386	78	80
data_fetch.py	72	1	99
mvo.py	27	0	100
tests/test_data_fetch.py	183	1	99
tests/test_ga1.py	242	4	98
tests/test_ga2.py	337	3	99
tests/test_ga3.py	210	3	99
tests/test_mvo.py	93	8	91
TOTAL	2120	116	95

Table 4.1: Test Coverage Report

The testing process involved both unit tests for individual functions and integration tests that verified the correct interaction between components. For example, we thoroughly tested the `data_fetch.py` file to ensure it handled API failures gracefully and properly cached results, while our genetic algorithm tests verified that the optimisation processes converged to meaningful solutions. Mock objects and patch decorators were extensively used to isolate components during testing, allowing us to validate behaviour without relying on external services or introducing non-deterministic results. Through comprehensive testing, we have developed a robust and reliable portfolio optimisation framework that can be trusted to perform consistently under various conditions, providing a solid foundation for financial decision-making.

Chapter 5

Results, Analysis & Evaluation

In this chapter we focus on a comprehensive analysis of our genetic algorithm-based portfolio optimisation implementations. We systematically evaluate them by explaining the experimental setups and evaluation metrics for each GA, and analyse the results accordingly

We calculate a deterministic MVO maximum Sharpe ratio using our MVO.py and stocks.csv file that returns the deterministic maximum value when given set of stock data on input. The implementation for this file has not been explored due to its minor relevance in the grand scheme of things for evaluation purposes, but is visible in the codebase.
The **stocks.csv** file existent in the codebase is a list of 25 diversified stocks across different sectors, and is the main data source we use for this evaluation section

5.1 GA1 Results

This section presents the results from the baseline Genetic Algorithm (GA1) designed for single-objective portfolio optimisation. The primary objective of this setup is to maximise the Sharpe ratio and to assess the feasibility of using a GA-based approach by comparing its performance to the deterministic convex Markowitz (MVO) solution, which produced a fixed optimal Sharpe ratio of **1.2505** for our example stocks.csv data file.

Experimental Setup

Two sets of GA parameter configurations were tested to evaluate the impact of parameter tuning on convergence and solution quality. Each configuration was run 15 times:

- **Configuration A:** 100 generations, population size of 60, mutation rate = 0.01

- **Configuration B:** 100 generations, population size of 20, mutation rate = 0.01

The following evaluation metrics were used:

1. **Best Sharpe Ratio Achieved:** The maximum Sharpe ratio obtained in a run.
2. **Convergence Behaviour:** The evolution of the best Sharpe ratio across generations.
3. **Accuracy to MVO Benchmark:** The deviation of the GA-derived Sharpe ratio from the known deterministic benchmark.
4. **Consistency and Reliability:** Measured via the average Sharpe ratio and standard deviation across the 15 runs.

5.1.1 GA1 Results: Configuration A

Best Sharpe Ratio	Average Sharpe Ratio	Standard Deviation
1.2405	1.2307	0.0079

Observations:

- The best Sharpe ratio of **1.2405** is approximately **0.80%** below the MVO benchmark of **1.2505**, indicating a strong approximation.
- The average Sharpe ratio of **1.2307** reinforces this, falling just **1.58%** short of the benchmark.
- The convergence graph for Configuration A exhibits a thicker and smoother trend, suggesting steady and reliable convergence across generations.

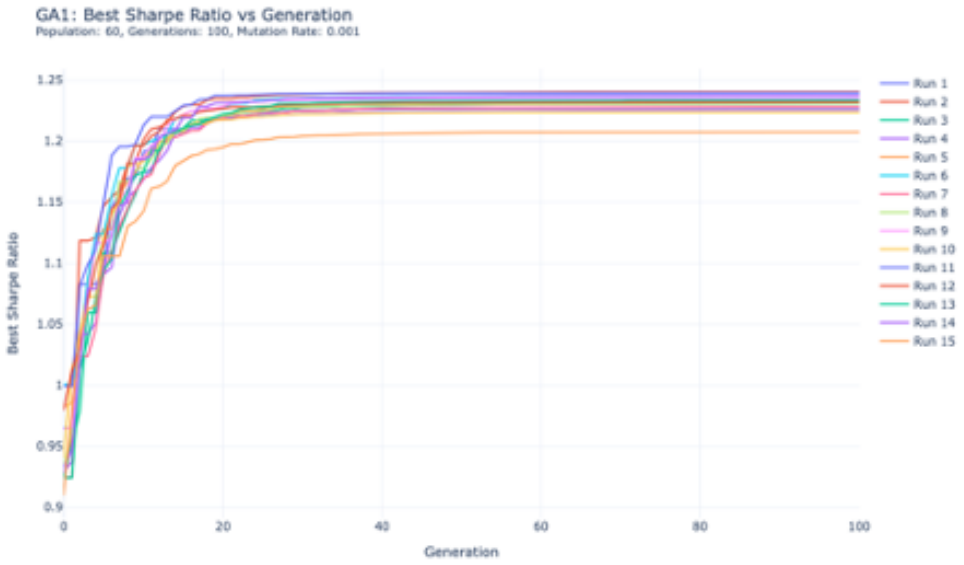


Figure 5.1: GA1 with Configuration A

5.1.2 GA1 Results: Configuration B

Best Sharpe Ratio	Average Sharpe Ratio	Standard Deviation
1.2057	1.1629	0.0351

Observations:

- The best Sharpe ratio of **1.2057** is approximately **3.58%** below the MVO benchmark of **1.2505.**
- The average Sharpe ratio of **1.1629** is notably weaker, deviating by around **7.01%** from the target value.
- The higher standard deviation of **0.0351** indicates significant variability across runs. This is reflected in the convergence graph below, which shows a less consistent trend than Configuration A.

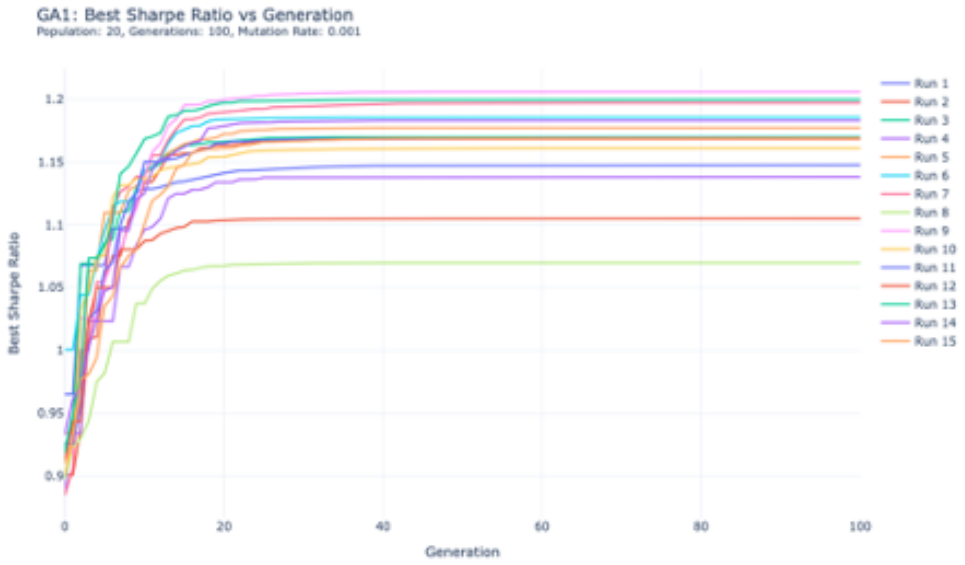


Figure 5.2: GA1 with Configuration B

GA1 Analysis

Benchmarking

- Configuration A achieved a Sharpe ratio very close to the MVO target (1.2328 vs. 1.2505), demonstrating the effectiveness of the GA in approximating the optimal solution.
- Configuration B underperformed significantly, with both lower best and average Sharpe ratios. The larger deviation and higher variance suggest that a smaller population limits the GA's ability to explore the solution space thoroughly, leading to premature convergence, as evidenced in the convergence graph.

Convergence Behaviour

- Configuration A displayed smoother and more consistent convergence patterns, evidenced by a thicker and more stable trend line across generations.
- In contrast, Configuration B showed erratic convergence, indicating instability and early convergence due to the smaller population size.

Parameter Sensitivity

- The only parameter difference between both configurations was the population size (60 in A vs. 20 in B).

- The stark contrast in outcomes underscores the importance of a sufficiently large population for:
  - Enhancing diversity and expanding the solution space.
  - Improving stability and reducing the variability of results.

This analysis confirms that while the GA approach cannot outperform the deterministic MVO benchmark, it can approximate it closely and consistently when well-configured. These findings validate the use of GAs in later setups and highlight their reliability in solving complex portfolio optimisation problems.

5.2 GA2 Results

This section presents the outcomes of our bi-objective portfolio optimisation using NSGA-II (GA2). In this setup, the genetic algorithm is designed to simultaneously minimise risk (volatility) and maximise expected return (expressed as a negative objective for optimisation purposes), resulting in a Pareto frontier of optimal trade-offs.

The best portfolio is selected from this frontier based on its Sharpe ratio (or user preference, as all portfolios on the frontier are efficient). This is then compared against the deterministic Markowitz benchmark Sharpe ratio of **1.2505** established in the previous setup.

Experimental Setup

To evaluate the impact of the number of generations on the quality and diversity of the Pareto frontier, two configurations were tested, each run 15 times. Population size and mutation rate were held constant:

- **Configuration A:** 10 generations, population size of 100, mutation rate = 0.1
- **Configuration B:** 200 generations, population size of 100, mutation rate = 0.1

The following evaluation metrics were considered:

1. **Pareto Frontier Diversity:** Number of candidate portfolios on the final frontier.

2. **Best Portfolio Selection:** Highest Sharpe ratio achieved among all solutions.

3. **Convergence Behaviour:** How the frontier evolves and densifies over generations.

4. **Growth Pattern:** Rate at which frontier solutions increase in early generations.

5.2.1 GA2 Results: Configuration A (10 generations)

Observations:

- **Frontier Diversity:** The final Pareto frontier is sparse, containing significantly fewer candidate portfolios compared to longer runs.
- **Best Portfolio Sharpe Ratio:** The best-performing portfolio achieved a Sharpe ratio of **1.1763**, which is approximately **5.94%** below the MVO benchmark of **1.2505**.
- **Frontier Growth:** The number of solutions on the frontier more than doubled between generation 5 and generation 10. This demonstrates a rapid early growth rate in solution diversity, with the first **10%** of generations contributing disproportionately to the total frontier expansion.

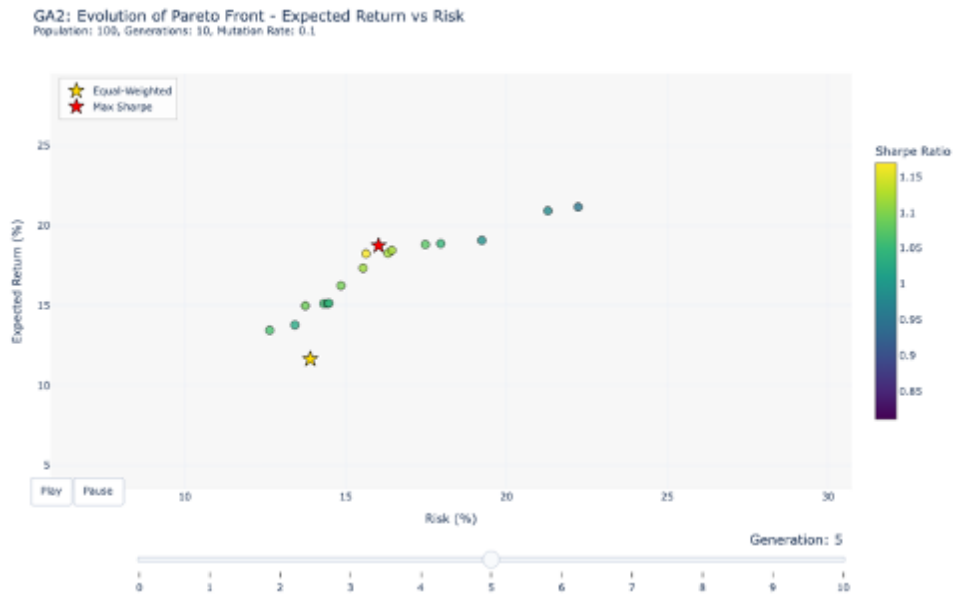


Figure 5.3: GA2, Configuration A at generation 10



Figure 5.4: GA2, Configuration A at generation 5

5.2.2 GA2 Results: Configuration B (200 generations)

Observations:

- **Frontier Diversity:** A larger and more diverse Pareto frontier was generated, offering a broad range of risk-return trade-offs.
- **Best Portfolio Sharpe Ratio:** The highest Sharpe ratio achieved was 1.2410, only 0.76% below the MVO benchmark of 1.2505 - indicating a high-performing evolutionary setup.
- **Frontier Growth Rate:** The number of candidate portfolios on the Pareto frontier quickly saturated, reaching the maximum count of 200 relatively early in the process, with slower growth thereafter.



Figure 5.5: GA2, Configuration B at generation 100

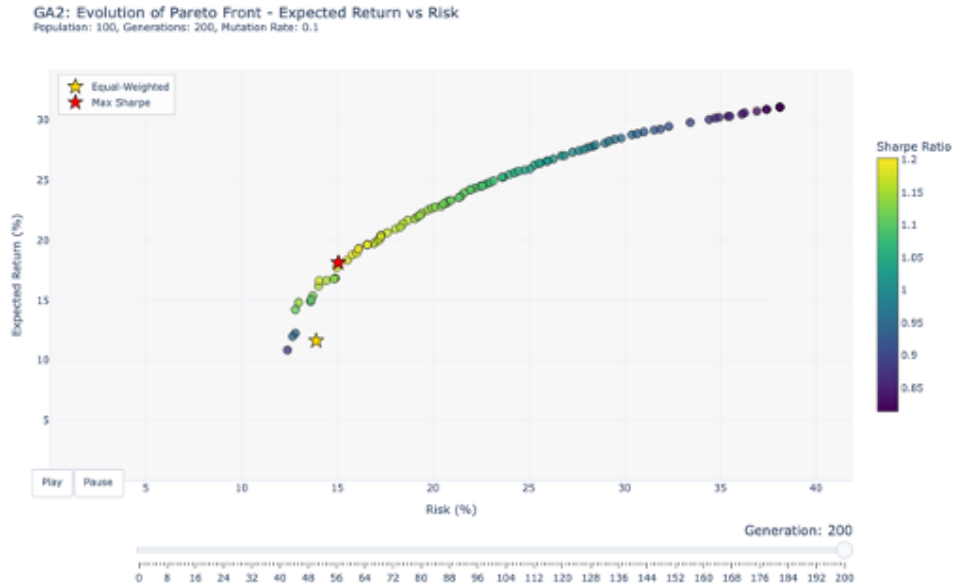


Figure 5.6: GA2, Configuration B at generation 200

GA2 Analysis

Benchmarking

- The best-performing portfolio from Configuration B achieved a Sharpe ratio of 1.2410, outperforming Configuration A's 1.1763. The proximity to the MVO benchmark (1.2505)

illustrates how increased generations help the algorithm better approximate the optimal trade-off between return and risk.

Convergence Behaviour

- Both configurations exhibit rapid initial convergence, but Configuration B displays continued growth and refinement in the Pareto frontier, resulting in a smoother and more stabilised frontier by generation 200.
- In contrast, Configuration A's frontier remains sparse, suggesting that while early improvements occur, fewer generations restrict the algorithm's ability to fully explore and exploit the solution space.

In summary, our evaluation of GA2 reveals that increasing the generation count significantly enhances both the diversity and quality of the Pareto frontier. Portfolios evolved with longer training achieve Sharpe ratios much closer to the deterministic MVO benchmark. While this benchmark remains aspirational, GA2 demonstrates that with sufficient parameter tuning, especially in terms of generation count, evolutionary approaches can robustly approximate optimal portfolio configurations. These findings build upon those from GA1, reinforcing the importance of tuning both population size and generation count. This understanding now forms the foundation for advancing to our final multi-objective optimisation in Setup 3.

5.3 GA3 Results

This section presents the outcomes of our tri-objective portfolio optimisation using NSGA-II (GA3). Unlike the single-objective setup in Section 5.1, GA3 simultaneously minimises portfolio risk (annual volatility), maximises expected return, and considers a cardinality constraint. The resulting set of non-dominated solutions forms a three-dimensional Pareto frontier, highlighting the trade-offs between return, risk, and sparsity. Performance is benchmarked against the deterministic Markowitz (MVO) Sharpe ratio of **1.2505**, while the influence of different parameter configurations, specifically cardinality thresholds, is analysed to assess solution quality and diversity.

Experimental Setup

To evaluate the algorithm both qualitatively and quantitatively, we use:

- **Dataset 1 (stocks.csv):** The same 25-stock dataset from previous setups.
- **Dataset 2 (stocks2.csv):** An expanded 50-stock dataset, enabling deeper exploration of the search space.

We also test two configurations based on the cardinality threshold parameter, exclusive to this setup:

- **Configuration A:** Cardinality threshold of 0.1
- **Configuration B:** Cardinality threshold of 0.0001

Population size is fixed at 300, the number of generations at 100, and the mutation rate is adaptive - another feature exclusive to our GA3 algorithm.

Evaluation Metrics

1. **Pareto Frontier Diversity:** Number and distribution of candidate portfolios across the risk-return-cardinality space.
2. **Cardinality Impact Analysis:** Influence of threshold values on portfolio selection and its relationship to risk-return objectives.
3. **Optimal Portfolio Selection:** Identification of best-performing portfolios at each cardinality level.

5.3.1 GA3 Results: Dataset 1 (stocks.csv), Configuration A (0.1)

The 3D Pareto frontier reveals distinct relationships between return, risk, and cardinality. Key observations include:

- **Frontier Structure:** The Pareto frontier displays clearly separated bands by cardinality levels, appearing as isolated lines in 3D space with significant vertical gaps.
- **Risk-Return Slices:** Within each cardinality level, portfolios tend to align in a nearly linear pattern in the risk-return plane, with few deviations.
- **Cardinality Interpretation:** No clear trend emerges between cardinality and risk, due to the sparse distribution of solutions.

- **Risk-Return Trade-off:** The classical inverse relationship between risk and return remains observable and consistent across cardinality layers, similar to findings in GA2.

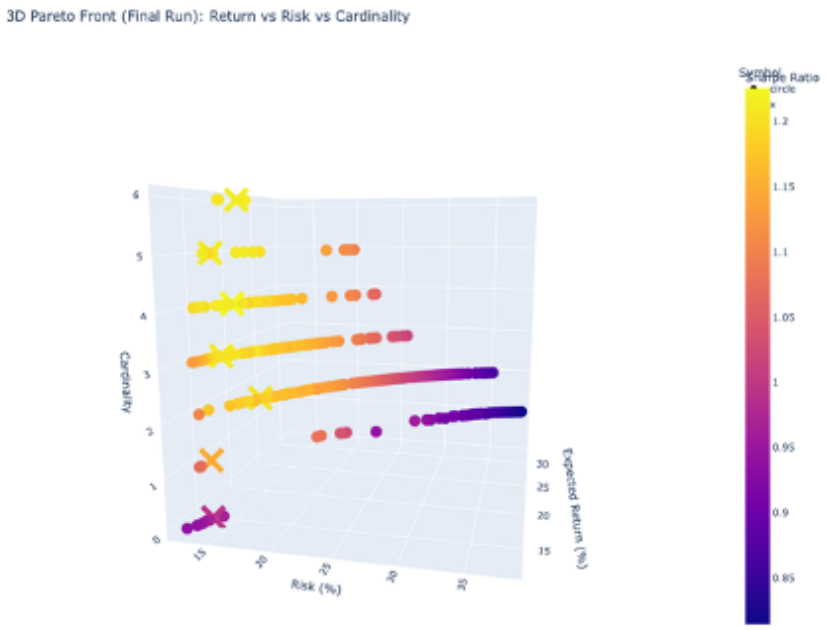


Figure 5.7: GA3 frontier with dataset 1 with Configuration A (1)

5.3.2 GA3 Results: Dataset 1 (stocks.csv), Configuration B (0.0001)

Compared to Configuration A, the results under a much lower threshold reveal several important distinctions:

- **Frontier Density:** The Pareto frontier is substantially denser and more populated with efficient portfolios, leading to better coverage of the search space.
- **Emerging Trends:** Clear relationships emerge: increasing cardinality is generally associated with lower risk, while portfolios with higher expected return tend to have lower cardinality.
- **Granularity:** The higher number of candidate solutions allows for a finer analysis of the risk-return-cardinality relationship.
- **Smooth Transitions:** Transitions between cardinality levels appear smooth and continuous, without the distinct gaps observed in Configuration A.

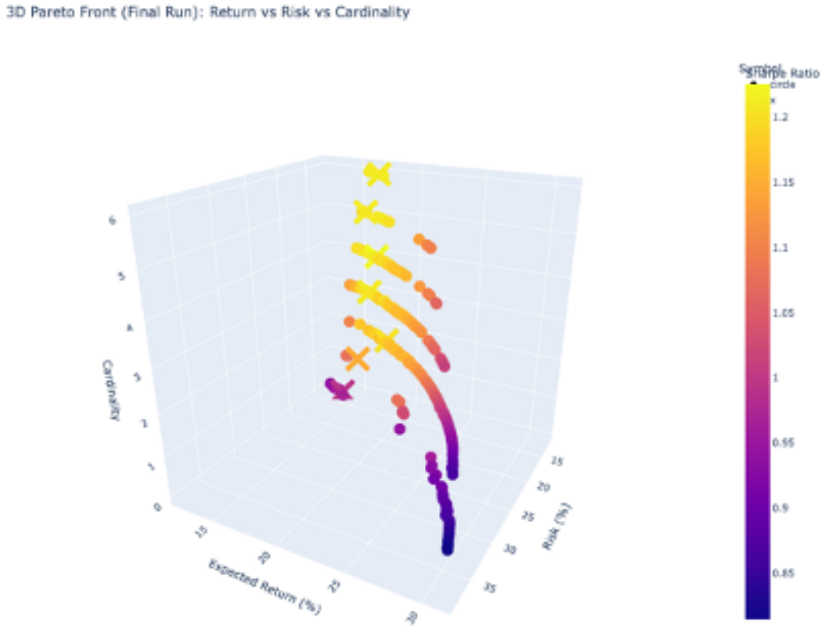


Figure 5.8: GA3 frontier with dataset 1 with Configuration B (2)

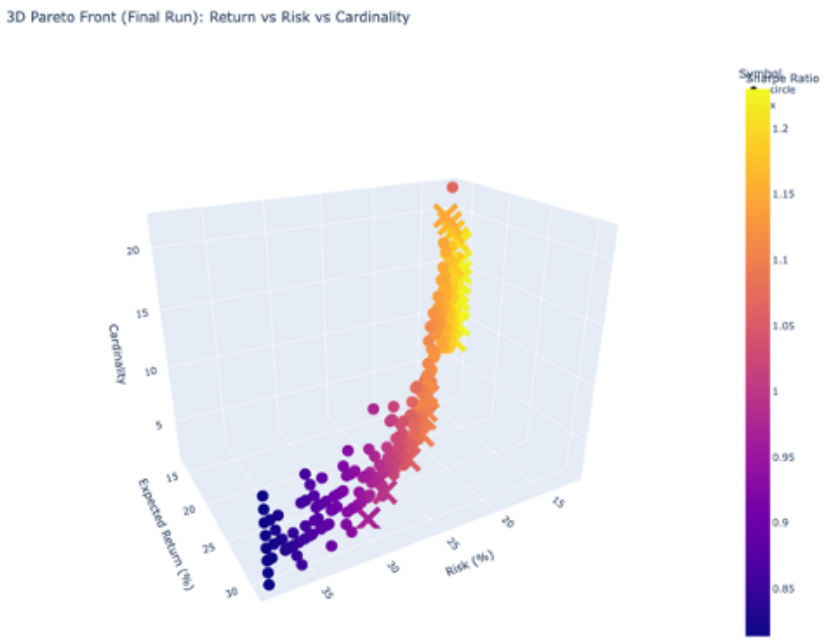


Figure 5.9: GA3 frontier with dataset 1 with Configuration B (1)



Figure 5.10: GA3 frontier with dataset 1 with Configuration B (2)

5.3.3 GA3 Results: Dataset 2 (stocks2.csv), Configuration A (0.1)

The expanded dataset of 50 stocks increases the complexity of the problem space and reveals more nuanced structural behaviours in both configurations.

- **Cardinality Structure:** Unlike Dataset 1, this configuration produces more defined lines along the cardinality axis.
- **Exploration Coverage:** Minimal whitespace appears within each cardinality level, showing improved exploration and solution density.
- **Frontier Clarity:** Despite the higher threshold, the risk-return frontier appears more well-defined, indicating the GA is capable of identifying competitive trade-offs within a broader solution space.

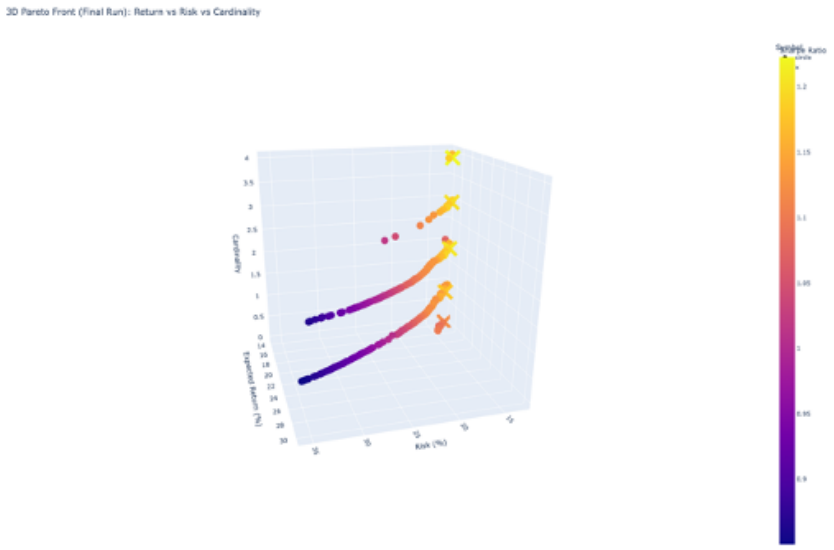


Figure 5.11: GA3 frontier with dataset 2 with Configuration A

5.3.4 GA3 Results: Dataset 2 (stocks2.csv), Configuration B (0.0001)

- **Boomerang Structure:** The optimal frontier forms a distinctive "boomerang" shape in 3D space, which is the most comprehensive structure observed across all setups.
- **Frontier Divergence:** From the midpoint of the cardinality axis, optimal portfolios diverge in both return and risk directions.
- **3D Coverage:** The frontier fills the 3D solution space more completely than any prior configuration, maintaining the risk-return trade-off across a wide range of cardinality values.
- **Interpretability:** This structure suggests that medium-cardinality portfolios may hold key positions in the trade-off between diversification, risk, and return.

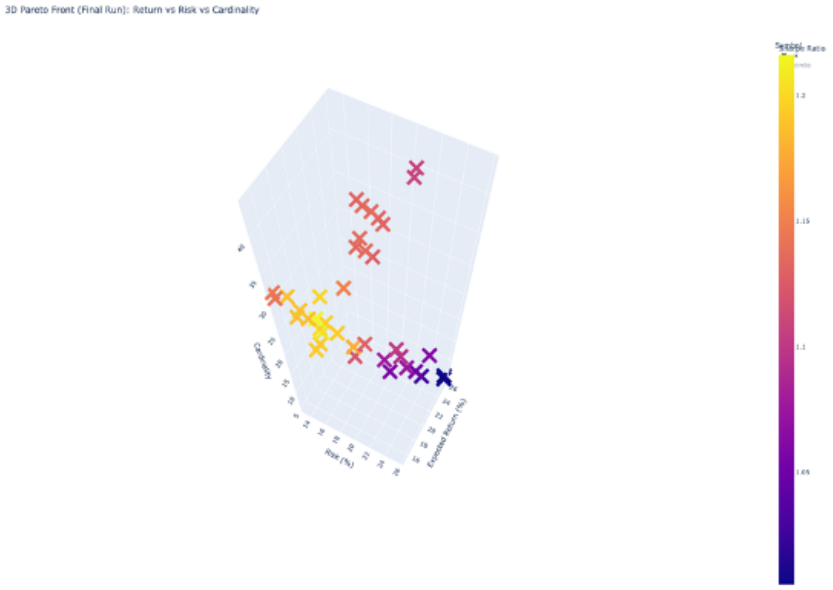


Figure 5.13: GA3 frontier with dataset 2 with Configuration B (2)

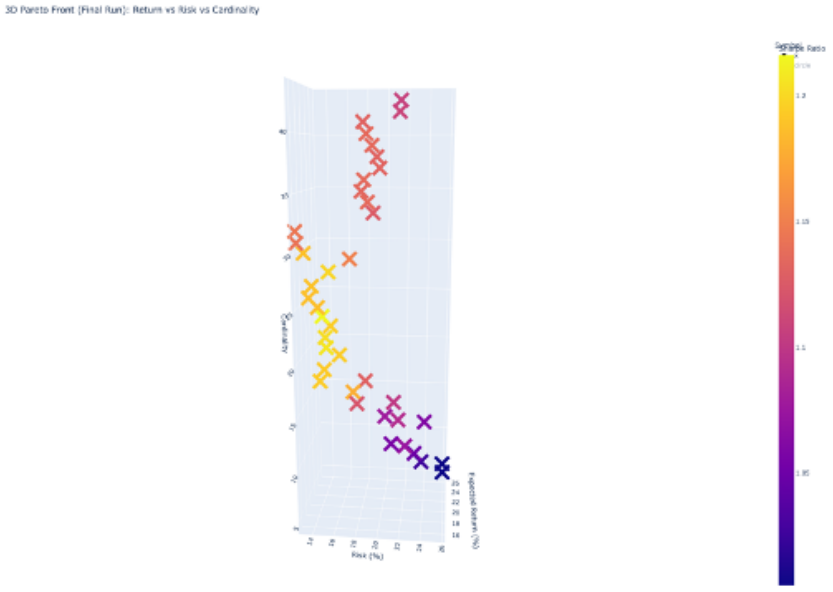


Figure 5.12: GA3 frontier with dataset 2 with Configuration B (1)

5.3.5 Quantitative Impact of Cardinality Threshold

The *cardinality threshold* parameter plays a crucial role in defining the effective search space and shaping the optimisation dynamics. For the 25-stock dataset:

- **Configuration B (threshold = 0.0001):** With such a low threshold, any non-zero weight is considered "active", allowing exploration of virtually the entire power set of stocks (excluding the empty portfolio). This results in:

$$2^{25} - 1 = 33{,}554{,}431 \text{ possible combinations}$$

- **Configuration A (threshold = 0.1):** Higher thresholds reduce the number of portfolios where a stock is considered active. Limiting to portfolios with at most 10 active stocks gives:

$$\sum_{k=1}^{10} \binom{25}{k} \approx 7{,}122{,}343 \text{ combinations}$$

For the expanded 50-stock dataset, the combinatorial impact becomes even more significant:

- **Configuration B (threshold = 0.0001):**

$$2^{50} - 1 = 1{,}125{,}899{,}906{,}842{,}623 \text{ combinations}$$

- **Configuration A (threshold = 0.1):**

$$\sum_{k=1}^{10} \binom{50}{k} \approx 10{,}329{,}144{,}451 \text{ combinations}$$

This contrast highlights how the cardinality threshold directly governs the complexity of the search space. Lower thresholds promote greater exploration at the cost of higher computational burden, while higher thresholds focus the search more narrowly.

GA3 Analysis

The GA3 results reveal consistent and interpretable trends in the relationship between cardinality, risk, and return:

1. **Low-cardinality portfolios (1–5 active stocks):**

   These portfolios exhibit the highest potential returns but also the highest risk. They show wide variability in performance and may reflect highly concentrated or sector-specific bets.

2. **Medium-cardinality portfolios (6–15 active stocks):**

   Portfolios in this range tend to achieve the highest Sharpe ratios, balancing risk and return effectively. They offer a practical middle ground between diversification and performance, suitable for investors with moderate risk tolerance.

3. **High-cardinality portfolios (16+ active stocks):**

   These portfolios have the lowest levels of risk but also lower expected returns. They demonstrate greater stability and may resemble passive investment strategies or index-like allocations.

In particular, the 50-stock dataset reveals a notable insight: optimal portfolios (those maximising the Sharpe ratio) consistently include between 8 and 12 active stocks, regardless of the threshold. This suggests the presence of a "sweet spot" of diversification, where risk is meaningfully reduced without significantly diluting returns through over-diversification.

Overall, GA3 effectively captures the complex trade-offs between return, risk, and cardinality. The resulting 3D Pareto frontiers provide a powerful visual and analytical framework for investors to select portfolios aligned with their preferences.

Furthermore, the inclusion of cardinality *as an optimisation objective* (rather than a hard constraint) represents a significant advance over traditional optimisation approaches. This allows for a broader and more flexible range of solutions, enhancing personalisation for individual investor needs and constraints.

5.4 Evaluation vs. Requirements

In this section, we evaluate each category of requirements (Functional, Non-Functional, and Optional) against our three main genetic algorithm setups: **GA1 (Single-Objective Sharpe Ratio)**, **GA2 (Bi-Objective NSGA-II)**, and **GA3 (Tri-Objective NSGA-II with Fama-French + PSO Integration)**.

Table 5.1: Evaluation of Functional Requirements Across GA Implementations

Req.	**Description**	**GA1 Passed?**	**GA2 Passed?**	**GA3 Passed?**
FR1	Implement a genetic algorithm (GA) based portfolio optimisation system capable of processing a defined set of U.S. equities and evolving their allocations over multiple generations.	Yes	Yes, extended to NSGA-II	Yes, includes multi-objective and PSO elements
FR2	Integrate a robust fitness function that evaluates at least one industry-used portfolio optimisation goal (e.g., the Sharpe ratio function). [6]	Yes, Sharpe ratio maximised	Yes, risk vs. return objectives still included Sharpe	Yes, tri-objective still includes Sharpe ratio logic as part of return measure
FR3	Produce a Sharpe ratio for a given set of tickers that ideally is within 5–10% of the deterministic convex MVO solution [1] for the same set of tickers.	Yes, within 0.80% of MVO benchmark	Yes, within 0.76% of MVO benchmark	Yes, within 1.10% of MVO benchmark

Continued on next page

Table 5.1 – Continued from previous page

Req.	Description	GA1 Passed?	GA2 Passed?	GA3 Passed?
FR4	Provide visualisations to witness the GA's final Pareto front showing the trade-off between risk and returns, using Python libraries such as Matplotlib and/or Plotly.	Partially (single-objective convergence plot)	Yes, 2D Pareto front (risk vs. return)	Yes, 3D Pareto front (risk, return, cardinality)
FR5	Implement a tri-objective genetic algorithm (GA3) that simultaneously optimises for return, risk, and cardinality, producing a comprehensive 3D Pareto front visualisation.	Not applicable in GA1	Not applicable in GA2	Yes, tri-objective approach with cardinality
FR6	Integrate the Fama-French factor model in GA3 to generate expected returns based on systematic risk factors, replicating institutional-grade portfolio construction approaches.	No	No	Yes, used Fama-French factors for expected returns
FR7	Successfully implement PSO-inspired directional refinement post-mutation to improve convergence while maintaining genetic diversity across the solution space. [11]	No	No	Yes, integrated post-mutation PSO steps
FR8	Develop an adaptive mutation rate mechanism that dynamically adjusts based on population diversity metrics and convergence patterns.	No	Partially (fixed rate)	Yes, adaptive mechanism introduced

Continued on next page

Table 5.1 – Continued from previous page

Req.	Description	GA1 Passed?	GA2 Passed?	GA3 Passed?
FR9	Process and analyse at least 25 U.S. equities using stock data from the Alpha Vantage API spanning from January 2000 through December 2024.	Yes, same data pipeline used	Yes, same data pipeline used	Yes, same data pipeline used
FR10	Implement cardinality constraints allowing selection of up to 12 stocks from the universe, with appropriate penalty functions to enforce this constraint effectively.	No	No	Yes, cardinality constraint integrated

5.4.1 Non-Functional Requirements Evaluation

Table 5.2: Evaluation of Non-Functional Requirements Across GA Implementations

Req.	Description	GA1 Passed?	GA2 Passed?	GA3 Passed?
NF1	Outputs should be clearly presented both in the terminal and via interactive graphs using Python libraries such as Matplotlib or Plotly, displaying key metrics where possible (Sharpe ratio, expected return, risk, and cardinality) with appropriate colour schemes and legends.	Yes (terminal + basic Sharpe ratio plot)	Yes (terminal + 2D Pareto front)	Yes (terminal + 3D Pareto front + summary metrics)

Continued on next page

Table 5.2 – Continued from previous page

Req.	**Description**	**GA1 Passed?**	**GA2 Passed?**	**GA3 Passed?**
NF2	Complete algorithm execution for 100 generations with a population size of 100 chromosomes should complete within 10 minutes on standard research hardware (i5 CPU, 16GB RAM), with performance scaling appropriately with increased population sizes.	Yes (single-objective is much less complex)	Yes	Yes
NF3	The program must allow users to: select stocks by entering tickers, and adjust all key algorithm parameters (population size, generations, crossover/mutation rates).	Partial	Yes	Yes (extended interface for tri-objective and advanced parameters)
NF4	Maintain a modular architecture that clearly separates concerns between algorithm, data processing, and visualisation components.	Yes (modular design implemented)	Yes (distinct modules for NSGA-II)	Yes (integrated modules remain consistent)
NF5	The system should incorporate at least two advanced evolutionary techniques (adaptive mutation, elitism, tournament selection, or particle swarm hybridisation) with improvement in solution quality over basic implementations.	Partially (tournament selection only)	Yes (elitism + tournament selection)	Yes (elitism + tournament + adaptive mutation + particle swarm hybridisation)
NF6	Provide comprehensive error handling and user feedback, with meaningful error messages for API failures, data inconsistencies, and algorithm convergence issues.	Partially (basic logging)	Yes	Yes (+ advanced logs)

Table 5.3: Evaluation of Optional Requirements Across GA Implementations

Req.	**Description**	**GA1 Passed?**	**GA2 Passed?**	**GA3 Passed?**
OR1	Caching mechanism for stock data to accelerate repeated runs and improve development and testing efficiency, reducing API calls by at least 90% for subsequent executions.	Via data_fetch.py	Via data_fetch.py	Via data_fetch.py
OR2	Playback visualisation of the algorithm process (e.g., dynamic Pareto frontier updates) to offer immediate insight into convergence and stability across generations.	No	Yes (visualise evolution of Pareto frontier)	No (too computationally heavy)
OR3	Integration of additional portfolio performance metrics beyond Sharpe ratio, such as Sortino ratio, maximum drawdown, and Value-at-Risk (VaR) calculations to provide comprehensive risk assessment.	No	Partially	Yes (Sortino, Sd & max drawdown)
OR4	Implementation of alternative evolutionary approaches (Differential Evolution, CMA-ES) as comparative benchmarks to evaluate the relative performance of our hybrid GA-PSO implementation.	No	No	No

5.5 Comparative Analysis

Table 5.4: Comparative Analysis of Genetic Algorithm Portfolio Optimisation Implementations

Feature/Component	**GA1.py**	**GA2.py**	**GA3.py**
Optimisation Objective	Single-objective (maximise Sharpe ratio)	Multi-objective (maximise return, minimise risk)	Multi-objective with cardinality (maximise return, minimise risk, minimise number of assets)
Algorithm Type	Basic genetic algorithm	NSGA-II (Non-dominated Sorting Genetic Algorithm II)	NSGA-II with PSO hybrid and adaptive mutation
Selection Method	Tournament selection (tournsize=2)	Tournament selection (tournsize=3)	Tournament selection (tournsize=3)
Crossover Method	Uniform crossover (indpb=0.5)	Simulated Binary Crossover (SBX)	Simulated Binary Crossover (SBX)
Mutation Method	Simple random value mutation	Simple random value mutation	Gaussian mutation with adaptive rate
Expected Return Calculation	Historical mean returns	Historical mean returns	Fama-French three-factor model
Diversity Management	None	None	Adaptive mutation rate based on population diversity
Local Search	None	None	Particle Swarm Optimisation (PSO) for local refinement
Population Size	Default 100	Default 100	Default 300
Visualisation	Basic convergence graph	Animated Pareto front evolution	3D Pareto front with interactive plots

Continued on next page

Table 5.4 – continued from previous page

Feature/Component	**GA1.py**	**GA2.py**	**GA3.py**
Performance Metrics	Basic (return, risk, Sharpe)	Expanded (includes Pareto front growth)	Expanded
Sensitivity Analysis	Simple multi-run option with visualisation	Multi-run with statistics	Multi-run with statistics
Repair Mechanism	Normalise weights to sum to 1	Normalise weights to sum to 1	Normalise weights to sum to 1
Constraints Handling	Simple sum-to-one	Simple sum-to-one	Sum-to-one plus cardinality objective
UI/Reporting	Basic Rich tables	Enhanced Rich tables with panel displays	Advanced Rich tables with live updates

Key Evolution Points:

- **GA1 → GA2**: Evolution from single-objective to multi-objective optimisation using NSGA-II, introducing the concept of Pareto-optimal portfolios.
- **GA2 → GA3**:
  - Addition of cardinality as a third objective
  - Integration of advanced financial models (Fama-French)
  - Incorporation of hybrid optimisation with PSO local search
  - Introduction of adaptive mutation rates based on population diversity
  - More comprehensive performance metrics and visualisation capabilities

Chapter 6

Legal, Social, Ethical & Professional Issues

This chapter evaluates potential risks and considerations arising from the use of evolutionary algorithms in financial applications. It explores connections to regulatory compliance, social impact, and ethical practices, particularly within the context of academic research.

6.1 Legal Issues

This project utilises only publicly available financial data sources, namely the Alpha Vantage API and the Fama-French factor datasets. The `data_fetch.py` script is configured to access data through a student API key, ensuring all data collection remains within the public domain. No personal or proprietary data is used at any stage, ensuring compliance with GDPR and other data protection frameworks.

Furthermore, the system is strictly intended for research purposes and is not deployed in any real-time trading or financial institution environment. As such, it does not trigger any obligations under financial regulatory bodies such as the FCA or SEC. All API usage adheres to published terms of service, supporting both transparency and legal compliance.

6.2 Social and Ethical Implications

Although the algorithms developed could theoretically inform automated trading strategies, their primary use case in this project is academic and exploratory. In real-world financial

systems, such tools could potentially exacerbate market inequalities by favouring institutions with access to superior data and computational resources.

In contrast, our approach uses only freely available data, which is likely already incorporated into market prices, particularly for liquid equities. This reduces the potential for unfair market exploitation. Nevertheless, overreliance on automated decision-making systems may contribute to displacement of roles in traditional financial sectors. These considerations emphasise the need for ethical oversight, especially when transitioning research systems into production.

6.3 Professional Issues

There exists a hypothetical risk that techniques showcased here could be misappropriated to gain an unfair edge in trading environments. However, the implementation is framed to encourage transparent, multi-objective optimisation that discourages reckless risk-seeking behaviour.

Adherence to professional codes of conduct, openness in method, and use of non-proprietary data all reflect the project's commitment to ethical, responsible research. It should also be acknowledged that real-world financial systems often rely on private, high-frequency data, such as that used in advanced models like the Barra Factor framework; giving them an edge that is inaccessible to academic researchers.

This distinction reinforces the project's goal: to foster accessible, well-governed research into portfolio optimisation, rather than to compete with institutional-grade systems.

Chapter 7

Conclusion

7.1 Summary of Key Findings

This project has demonstrated that evolutionary algorithms, particularly Genetic Algorithms (GAs) and a hybrid GA-PSO approach, are viable and scalable tools for portfolio optimisation tasks of increasing complexity.

The initial setup (GA1) applied a single-objective GA to approximate the deterministic Markowitz Mean-Variance Optimisation (MVO). Results confirmed that GAs can produce efficient frontiers comparable to classical optimisation under standard risk-return trade-offs.

Setup 2 (GA2) extended this to a bi-objective context using NSGA-II. The resulting Pareto frontiers effectively captured trade-offs between return and volatility, highlighting how evolutionary approaches can produce interpretable solutions under conflicting objectives.

Setup 3 (GA3) introduced a third objective: cardinality constraints, and incorporated factor-based returns derived from the Fama-French model. PSO-style velocity refinements were integrated to improve convergence. This setup more closely modelled real-world considerations such as diversification and sparse asset selection, demonstrating the flexibility of hybrid meta-heuristics.

Key findings across all setups include:

- Evolutionary algorithms are capable of generating high-quality, diverse portfolios even under complex multi-objective constraints.
- Population size and generation count were identified as critical hyperparameters influencing exploration and convergence.

- Incorporating domain-specific financial features (e.g., factor models, cardinality constraints) within the evolutionary framework improved the practical relevance of solutions.

7.2 Limitations

Despite its contributions, the project has several limitations:

- **Data resolution:** The use of monthly return data limits the applicability of findings to long-term strategies and excludes higher-frequency dynamics relevant to active or intraday trading.
- **Model assumptions:** Constraints common in practical portfolio construction such as transaction costs; maximum turnover; or regulatory and sector limits were not modelled. This limits the realism and practical relevance of the results to an extent, as real-world portfolios must operate within such constraints, which can significantly affect the feasibility, cost-efficiency, and compliance of any proposed allocation strategy.
- **Computational scalability:** As problem size increases, so does the computational cost. The stochastic nature of evolutionary algorithms makes them sensitive to hyperparameter tuning and susceptible to premature convergence without adequate resources.
- **Academic scope:** The findings are based on backtesting and simulations using publicly available datasets. Translating this work into production-grade systems would require validation against proprietary data, comprehensive risk models, and adherence to regulatory standards.
- **Stochastic Precision:** The inherent stochasticity of genetic algorithms makes it challenging to achieve precise investment decisions. This limitation suggests that such algorithms should only be used to guide general investment strategies rather than serve as the core decision-making factor. This is evidenced by the fact that no GA has ever reached, nor is expected to reach, the benchmark MVO Sharpe ratio value. Nonetheless, these algorithms provide qualitative insights into various trade-offs and investment perspectives that conventional convex optimisation methods might overlook.

7.3 Future Work

This work opens several avenues for further research:

- **Advanced risk modelling:** Incorporating multi-factor models such as Barra could improve granularity in risk decomposition and better align optimisation with institutional portfolio constraints.
- **Algorithmic hybridisation:** Further exploration of hybrid algorithms (e.g., combining GAs with Tabu Search, Simulated Annealing, or Reinforcement Learning) could enhance convergence speed and solution quality.
- **Scalability through quantum optimisation:** Mapping portfolio optimisation onto quantum-inspired frameworks such as Quantum Annealing may enable tractable exploration of large solution spaces, especially under discrete constraints.
- **Interactive decision tools:** Developing a user-facing optimisation tool or web interface based on this framework could support retail or institutional investors in navigating trade-offs among competing portfolio objectives.

This project establishes a strong foundation for applying evolutionary computation to financial portfolio optimisation, illustrating its adaptability in modelling real-world constraints and its promise for future multi-objective investment strategies.

References

[1] Harry Markowitz. Portfolio selection. *The Journal of Finance*, 7(1):77–91, 1952. doi: 10.2307/2975974.

[2] Portfolio optimization — wikipedia, the free encyclopedia. https://en.wikipedia.org/wiki/Portfolio_optimization#Correlations_and_risk_evaluation, 2024. [Online; accessed 3-April-2025].

[3] New research finds retail investing shift towards younger investors reshaping market trends. https://www.weforum.org/press/2025/03/new-research-finds-retail-investing-shift-towards-younger-investors-reshaping-market-trends/, 2025. [Online; accessed 3-April-2025].

[4] Khin Win and Rong Qu. *Hybrid Algorithm for Constrained Portfolio Selection Problem*. University of Nottingham, 2013.

[5] Francesco Cesarone, Antonio Scozzari, and Leone Tardella. Efficient algorithms for mean-variance portfolio optimisation with cardinality constraints. *Journal of Computational Finance*, 9(2):1–22, 2005. doi: 10.21314/JCF.2005.114.

[6] William F. Sharpe. Mutual fund performance. *Journal of Finance*, 19(3):425–442, 1966. doi: 10.2307/2977928.

[7] Victor DeMiguel, L. Garlappi, and R. Uppal. Optimal versus naive diversification: How inefficient is the 1/n portfolio strategy? *The Review of Financial Studies*, 22(5):1915–1953, 2009. doi: 10.1093/rfs/hhn070.

[8] Eugene F. Fama and Kenneth R. French. Common risk factors in the returns on stocks and bonds. *Journal of Financial Economics*, 33(1):3–56, 1993. doi: 10.1016/0304-405X(93)90023-5.

[9] John H. Holland. *Adaptation in Natural and Artificial Systems*. University of Michigan Press, Ann Arbor, MI, 1975. ISBN 978-0-472-08460-4.

[10] Marco Dorigo and Luca M. Gambardella. *Ant Colony System: A Cooperative Learning Approach to the Traveling Salesman Problem*. MIT Press, Cambridge, MA, 1997. ISBN 978-0-262-06139-9.

[11] J. Kennedy and R. Eberhart. Particle swarm optimization. In *Proceedings of IEEE International Conference on Neural Networks*, pages 1942–1948, 1995. doi: 10.1109/ICNN.1995.488455.

[12] Stylianos Kampakis. Are genetic algorithms for neural networks coming back? 2018. URL https://thedatascientist.com/genetic-algorithms-neural-networks/. Accessed: 2025-04-03.

[13] A.E. Eiben and J.E. Smith. *Introduction to Evolutionary Computing*. Springer, Berlin, Heidelberg, 2003. ISBN 978-3-540-24686-2.

[14] K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan. A fast and elitist multiobjective genetic algorithm: Nsga-ii. *IEEE Transactions on Evolutionary Computation*, 6(2):182–197, 2002. doi: 10.1109/4235.996017.

[15] Papers With Code. Genetic algorithms, 2019. URL https://production-media.paperswithcode.com/methods/gadiagram-300x196_jThbitI.png. Accessed: 2025-04-03.

[16] Ahmad B. A. Hassanat. Enhancing genetic algorithms using multi mutations. *International Journal of Computer Science and Information Security*, 4(1):25–30, 2006.

[17] MathWorks. Convex optimization, 2025. URL https://www.mathworks.com/discovery/convex-optimization.html. Accessed: 2025-04-03.

[18] Melanie Mitchell. *An Introduction to Genetic Algorithms*. MIT Press, Cambridge, MA, 1998. ISBN 978-0-262-12186-3.

[19] Stephen Boyd and Lieven Vandenberghe. *Convex Optimization*. Cambridge University Press, Cambridge, UK, 2004. ISBN 978-0521833783.

[20] Pat. The mystery of volatility estimates from daily versus monthly returns. *Portfolio Probe Blog*, 2011. URL https://www.portfolioprobe.com/2011/11/08/the-mystery-of-volatility-estimates-from-daily-versus-monthly-returns/.

[21] Firstname Jung and Firstname Lee. Market dynamics and inflation: A post-2000 perspective. *Journal of Financial Research*, XX(X):XX–XX, 2024. doi: 10.XXXX/XXXXX.

[22] NumPy Developers. Numpy, 2025. URL https://github.com/numpy/numpy. Accessed: 2025-04-03.

[23] Pandas Developers. pandas, 2025. URL https://github.com/pandas-dev/pandas. Accessed: 2025-04-03.

[24] DEAP Developers. Deap: Distributed evolutionary algorithms in python, 2025. URL https://github.com/DEAP/deap. Accessed: 2025-04-03.

[25] Plotly Developers. plotly.py: The interactive graphing library for python, 2025. URL https://github.com/plotly/plotly.py. Accessed: 2025-04-03.

[26] Python Software Foundation. argparse – parser for command-line options, arguments and sub-commands, 2025. URL https://docs.python.org/3/library/argparse.html. Accessed: 2025-04-03.

[27] Textualize Developers. Rich: Rich text and beautiful formatting in the terminal, 2025. URL https://github.com/Textualize/rich. Accessed: 2025-04-03.

[28] Unknown. Tournament selection diagram, 2025. URL https://miro.medium.com/v2/resize:fit:1400/1*ASSdFItRZZ9K6tpNkaa1_A.jpeg. Accessed: 2025-04-03.

[29] Stack Overflow Community. Simulated binary crossover (sbx) crossover operator example, 2014. URL https://stackoverflow.com/q/XYZ.

[30] Karishma Rahman and Clemente Izurieta. An approach to testing banking software using metamorphic relations. In *2023 IEEE 20th International Conference on Information Reuse and Integration for Data Science (IRI)*, pages 54–61. IEEE, 2023. doi: 10.1109/IRI58776.2023.00017.

Appendix A

User Guide

A.1 Instructions

This appendix provides a concise walkthrough on setting up and using the portfolio optimisation tools powered by genetic algorithms. The repository includes three main implementations:

- **GA1.py: Single-Objective** - Optimises for maximum Sharpe ratio.
- **GA2.py: Multi-Objective (NSGA-II)** - Balances return and risk, with options for an animated 2D Pareto front.
- **GA3.py: Advanced Multi-Objective with Cardinality Constraint** - Uses Fama-French factors to optimise return, risk, and portfolio cardinality, displaying a 3D Pareto front.

Setup

- Run the setup script to create a virtual environment and install dependencies:

```
chmod +x setup.sh
./setup.sh
```

Testing

- Execute unit tests and generate a coverage report with:

```
chmod +x tests.sh
./tests.sh
```

Data Requirements

- A CSV file (default: `data/stocks.csv`) is required, containing a date column in `YYYY-MM-DD` format and asset price data.

Usage

- **GA1:** Run for maximum Sharpe ratio:

```
python GA1.py -g
```

- **GA2:** Run NSGA-II for the return-risk tradeoff:

```
python GA2.py -g
python GA2.py -a   % For animated Pareto front evolution
```

- **GA3:** Run the advanced multi-objective GA:

```
python GA3.py -g
```

Additional Options

- **Multiple Runs:** For sensitivity testing:

```
python GA1.py -r 10 -g
python GA2.py -r 5 -g
python GA3.py -r 3 --aggregate-plot
```

- **GA3 Specific:** Set the minimum weight threshold for asset inclusion:

```
python GA3.py --cardinality-threshold 0.001 -g
```

Visualisations and Output

- Interactive plots (2D/3D) are generated using Plotly and open automatically in your web browser.
- Detailed performance metrics (e.g., portfolio weights, expected returns, volatility, risk-adjusted measures) are printed to the console.

Helper Scripts

- **data_fetch.py** - Fetches and caches stock data from the Alpha Vantage API.
- **mvo.py** - Performs standard Mean-Variance Optimisation to determine the maximum Sharpe ratio portfolio we use in our evaluations

Optional Coverage Report

- Generate an HTML report for detailed test coverage:

  ```
  coverage html
  ```

Note: More in-depth instructions can be found in the `README.md` file in the source code.

Appendix B

Source Code

Note: The source code has been uploaded via KEATS as a zip under the name of **YunusSufian [redacted] PRJ SourceCode.zip**.