

OTDK-dolgozat

2025

BUDAPESTI CORVINUS EGYETEM

PÉNZÜGY INTÉZET

**KARDINALITÁSKORLÁTOS PORTFÓLIÓOPTIMALIZÁCIÓ
EVOLÚCIÓS ALGORITMUSOKKAL**

**CARDINALITY-CONSTRAINED PORTFOLIO OPTIMISATION WITH
EVOLUTIONARY ALGORITHMS**

Péter Veronika

Konzulens: Czupy Gergely János

Kézirat lezárásának dátuma: 2023. 04. 17.

Table of contents

I. Introduction.....	1
II. Portfolio optimisation theory	3
II.1. The Markowitz portfolio optimisation problem	3
II.1.1. Diversification	3
II.1.2. Mean-Variance Analysis.....	5
II.1.3. The unconstrained optimisation problem.....	6
II.2. Improvements and current trends in the Markowitz model	7
II.2.1. Dynamic portfolio optimisation.....	7
II.2.3. Choice of decision criteria	8
II.2.4. Practical constraints	9
II.2.5. Cardinality constraints.....	10
II.3. Mixed integer programming.....	10
III. Introduction to evolutionary computing	12
III.1. Darwinian Evolution Theory	12
III.2. The four major Evolutionary Algorithm variants	13
III.3. Main components of Evolutionary Algorithms	13
III.3.1. Representation	14
III.3.2. Population	14
III.3.3. Evaluation function	15
III.3.4. Selection mechanism	15
III.3.5. Variation operators	16
III.3.6. Termination.....	16
III.4. Genetic Algorithms	17
III.4.1. The structure of genetic algorithms.....	17
III.4.2. Improvements to genetic algorithms	18
III.5. Advantages and disadvantages of evolutionary algorithms	19
IV. Methodology.....	21
IV.1. The unconstrained problem.....	21
IV.2. The genetic algorithm.....	22
IV.2.1. Representation and initialisation.....	23
IV.2.2. Evaluation function.....	24
IV.2.3. Parent selection	24
IV.2.4. Recombination and mutation.....	25
IV.2.5. Survivor selection and termination	26

IV.3. The MIQP algorithm	27
IV.4. Comparison of the two methods.....	28
IV.4.1. Problem setting.....	29
IV.4.2. Constructing hypotheses	30
IV.4.3. Statistical tests	30
IV.5. Testing with real-world data	31
IV.5.1. Problems with real data usage	31
IV.5.2. Calibration for real data testing	32
V. Discussion of empirical results.....	33
V.1. Discussion of optimal solution.....	33
V.1.1. Results from the Genetic Algorithm	33
V.1.2. Results from the MIQP solver	36
V.2. GA and MIQP comparison	37
V.2.1. Discussion of findings	38
V.2.2. Results of the hypothesis testing.....	39
V.3. Results of testing with market data	42
VI. Conclusion and future work	45
VII. Bibliography	47
Appendix 1.	52
Appendix 1.1 – Random correlation matrix generation	52
Appendix 1.2 – Define data generation function	52
Appendix 1.3 – Define unconstrained optimisation function	53
Appendix 1.4 – Define MIQP optimisation function	54
Appendix 1.5 – Define class that stores results	55
Appendix 1.6 – Define function to select parents and generate offspring	56
Appendix 1.7 – Define survivor selection	56
Appendix 1.8 – Define genetic algorithm.....	57
Appendix 1.9 – Code to run simulation	58
Appendix 2. Results of the Wilcoxon signed-rank test for simulated data set.....	60
Appendix 2.1. Right-tailed Wilcoxon signed-rank test.....	60
Appendix 2.1. Left-tailed Wilcoxon signed-rank test	62

List of Figures

Figure 1. Possible portfolio combinations from two securities for different values of correlation.	4
Figure 2. Illustration of the Markowitz Minimum-Variance Frontier with N risky assets.	6
Figure 3. Evolutionary Algorithm flowchart.	14
Figure 4. Algorithm: Unconstrained optimisation.	22
Figure 5. Algorithm: Genetic Algorithm.	23
Figure 6. Illustration of uniform crossover, where crossover rate = 0.5.....	25
Figure 7. Algorithm: Mixed Integer Quadratic Programming.	28
Figure 8. Comparison of average objective value achieved by the GA with 95% confidence interval for different portfolio sizes (K).	35
Figure 9. Comparison of average objective value with 90% confidence interval between each algorithm for different portfolio sizes (K) under strict time limit.	38
Figure 10. Comparison of average objective value with 90% confidence interval between each algorithm for different portfolio sizes (K) under broader time limit.....	39
Figure 11. Performance improvement of the GA for increasing problem size with 90% confidence interval.	41
Figure 12. Comparison of average objective value for real data set with 90% confidence for different time limits (T).....	42
Figure 13. Comparison of performance improvement of the GA in the simulated and real environment for increasing problem size with 90% confidence interval.	43

List of Tables

Table 1. The Markowitz Quadratic Optimisation problem with N risky assets.....	6
Table 2. Optimisation along the efficient frontier	7
Table 3. Average results of the Genetic Algorithm for N = 500.....	34
Table 4. Average results of the Genetic Algorithm for N = 5000.....	34
Table 5. Average results of the MIQP algorithm for N = 500.	36
Table 6. Average results of the MIQP algorithm for N = 5000.	37
Table 7. Results of the right-tailed Wilcoxon signed-rank test for N = 500.	40
Table 8. Results of the left-tailed Wilcoxon signed-rank test for N = 5000.	41
Table 9. Results of the left-tailed Wilcoxon signed-rank test with the maximum assets (K) to be chosen is 250 and the time limit (T) is 10.	44
Table 10. Results of the left-tailed Wilcoxon signed-rank test with the maximum assets (K) to be chosen is 250 and the time limit (T) is 60.	44

I. Introduction

Portfolio optimisation has been introduced to investors through the seminal paper of Harry Markowitz in 1952, conceptualising portfolio construction as an optimisation problem through the mean-variance framework. In the Markowitz model, the investor considered the return and risk of the portfolio simultaneously, aiming to find the optimal balance for the portfolio. The mean-variance model served as a highly effective method for quantifying the relationship between risk and return, offering investors a mathematical approach to diversification. The portfolio construction process thus turned into a problem that can be solved using quadratic programming, with the objective of minimising the risk of the portfolio for any given level of return.

Over the last 70 years, the Markowitz model has been a major area of interest with a growing body of literature dedicated to enhancing to the original framework. The addition of various investment preferences and practical constraints has increased the complexity of the optimisation problem substantially, necessitating the adoption of more advanced optimisation techniques. For instance, restricting the number of assets held in the portfolio (for example, which 100 assets should be included in the optimal portfolio from an investment universe of 5000 assets) transforms the original quadratic program into a mixed integer quadratic program, which classifies as a problem of exponential complexity.

While mixed integer programming is commonly used for solving portfolio optimisation problems that are subject to integer constraints, it is not effective in all investment environments. In particular, as the number of investable securities in the optimisation increases, the time and computational effort required by the program also grows exponentially, making it nearly impossible to find the global optimum. Therefore, in practical cases, stochastic optimisation methods such as evolutionary algorithms are also frequently used to solve these problems.

Evolutionary computing represents the most important subgroup of stochastic optimisation methods. The concept emerged from the late 1940s as a means of metaphorically applying the Darwinian evolution theory into the optimisation process. In evolutionary algorithms, a population representing possible solutions to a problem is evolving through subsequent generations to reach the optimum. As the method is drawing inspiration of the natural evolution process, the individuals in the population are chosen to become parents and generate offspring, with the better performing individuals receiving a higher chance to pass on their genetic code.

The goal of the algorithm is to improve the overall performance of the population by replacing poorly performing individuals with fittest members.

Genetic algorithms are a branch of evolutionary algorithms that are mainly characterised by their structure and terminology that have been adopted from the field of genetics. The individuals in genetic algorithms are encoded into binary strings of data called chromosomes. Through their binary representation, they are quite effective in exploring the search space of optimisation problems and are therefore quite popular among the different evolutionary algorithm variants.

Although the concept of evolutionary algorithms has been around for more than 70 years, the idea of applying it to portfolio optimisation is relatively new, hence the approach has received little attention in research literature. Therefore, the purpose of this paper is to analyse and review the performance of evolutionary algorithms, and in particular genetic algorithms for portfolio optimisation. The focus of research is the cardinality constrained portfolio optimisation problem, which means that there is a restriction on the number of assets allowed in the portfolio. The performance of genetic algorithms is compared to the more conventional solution through mixed integer programming for an increasing scale of problem sizes.

The paper is composed of five themed chapters. The first chapter provides an overview of the Markowitz portfolio theory along with its improvements and modern solution approaches such as mixed integer programming. Chapter Two is addressed to introducing evolutionary and genetic algorithms through their structure and main components. The third chapter is concerned with the methodology used for analysing the efficiency of genetic algorithms and presents the algorithms created for solving the portfolio optimisation problem. Chapter Four examines the results of the empirical analysis performed on a simulated as well as a real data set. Finally, Chapter Five presents the conclusions of the results and summarises the main research findings, including the limitations of the current study and the recommendations for future work.

II. Portfolio optimisation theory

In this section, I am going to provide a brief theoretical background on portfolio optimisation, and more specifically on the Markowitz mean-variance analysis. I will also include a review of improvements to the original Markowitz method, particularly with regards to practical constraints. Finally, I will describe the cardinality constrained portfolio optimisation problem and how mixed integer programming may be utilised to solve it.

II.1. The Markowitz portfolio optimisation problem

Modern Portfolio Theory (MPT) originated in 1952 when Harry M. Markowitz published his seminal paper ‘Portfolio Selection’ in the Journal of Finance. Contrary to previous research, Markowitz described the process of portfolio construction as an optimisation problem and proposed the ground-breaking concept that the investor should not only be concerned about expected returns but ought to factor in risk as well (Markowitz, 1952). According to Markowitz, a crucial element of creating a diverse portfolio is taking into consideration the so-called risk-return trade-off. The concept effectively helped investors understand that if they seek to increase their portfolio’s expected returns, they can only do so by taking on more risk as well. And vice versa, eliminating variance can solely be achieved by giving up some of the expected returns.

II.1.1. Diversification

When investing under uncertainty, diversification is a key aspect to creating a profitable portfolio, a concept that has been widely recognised by financial analysts. The term diversification is described by Bodie, Kane and Marcus (2014, p. 10) as “*many assets are held in the portfolio so that the exposure to any particular asset is limited*”. Achieving a diversified portfolio is possible through a variety of strategies such as putting money into different types of securities, investing internationally or choosing securities from different industries. For instance, a portfolio that contains a hundred distinctive technology stocks can hardly be considered as a diversified investment, whereas investing for example 40% in technology, 30% in energy, 20% in health care and 10% in real estate can save the portfolio from huge losses in case of a downturn in the technology industry, if the other industries are not affected.

The important breakthrough came from Markowitz (1952, 1959) and Roy (1952), who were the first to put diversification in mathematical terms. The scientific approach to diversification quickly replaced naïve diversification and allowed investors to optimise their portfolios through the risk-return trade-off. In his 1959 book ‘Portfolio Selection: Efficient Diversification of

Investments', Markowitz demonstrated further techniques for portfolio analysis and quantifying the inputs to portfolio construction. The risk of a portfolio under the framework of MPT is measured by the standard deviation of returns. The effect of diversification as described by Markowitz comes from the Variance Sum Law which states that the variance of the weighted sum of two variables can be calculated as follows:

$$\sigma_{A+B}^2 = w\sigma_A^2 + (1-w)\sigma_B^2 + 2w(1-w)\sigma_{A,B}$$

where w denotes the weight invested in A ; σ is the standard deviation and $\sigma_{A,B}$ represents the covariance between A and B . Using the association between covariance and correlation, the formula can be modified to contain the more easily interpretable correlation (ρ):

$$\sigma_{A+B}^2 = w\sigma_A^2 + (1-w)\sigma_B^2 + 2w(1-w)\rho\sigma_A\sigma_B$$

As shown in the above equation, correlation between security returns plays an important part in diversification. When looking at the feasible set of portfolios constructed from two risky assets illustrated on Figure 1, it becomes evident that the different values of correlation can create very different mean-variance combinations. Effectively, the higher the correlation between the two securities, the higher the variance of their sum. Therefore, if the securities are highly correlated, the risk of the portfolio can scarcely be eliminated, and in case of perfect correlation, putting the two securities in the same portfolio would have no diversification effect. Meanwhile, securities that are negatively correlated can eliminate much of the risk, owing to the fact that in case the returns of one would fall, the other would move up. The important message from the Markowitz framework is that when putting together a portfolio, the investor has to consider the impact of each security, specifically how the security co-moves with the others in the basket.

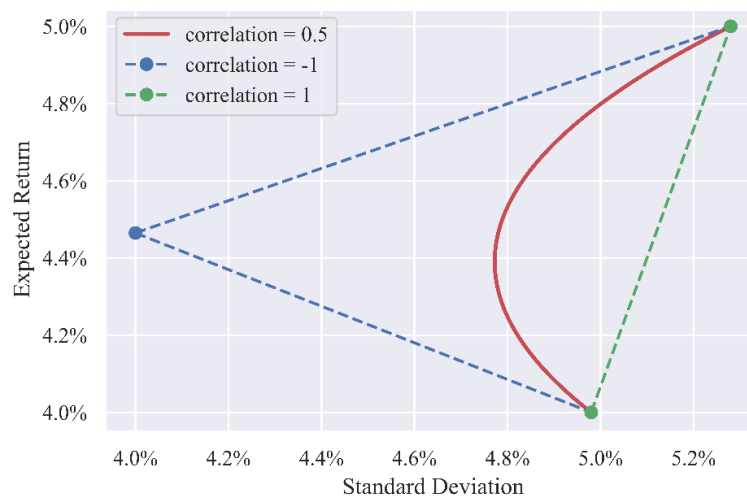


Figure 1. Possible portfolio combinations from two securities for different values of correlation. The figure has been created by the author.

II.1.2. Mean-Variance Analysis

The Mean-Variance Analysis proposed in Markowitz (1952, 1959) provided investors and financial analysts with a mathematical approach to portfolio allocation. His method has been widely discussed in academic literature. Merton (1972) provided an in-depth analysis of the efficient frontier with more than three assets, contributing useful inputs to the Mutual Fund Separation Theorem as well. Elton and Gruber (1997) reviewed MPT along with its developments and the most effective use cases for the present and the future. In a relatively recent research, Guo (2022) highlighted the advantages, as well as the limitations of the Markowitz model. Taken together, these studies support the significance of the Markowitz Mean-Variance model on portfolio optimisation and emphasize the advancements that were enabled by the original method. In addition to presenting the original Markowitz framework, the subsequent introduction to the Mean-Variance Analysis will also be following the contents of Fabozzi *et al.* (2007) and Prigent (2008), as these sources accurately introduce the enhancements to the original model as well.

Modern Portfolio Theory seeks to answer the question on how investors can find the optimal portfolio for a specific time horizon? The initial setting is the feasible set of assets with known expected return (r) and standard deviation (σ). The expected return of any particular security is given by the

$$\frac{\text{Expected price} - \text{Opening price} + \text{Dividends}}{\text{Opening price}}$$

formula for any given period. Risk is measured by the standard deviation of returns, which refers to the variability of returns. The risk-return trade-off is paired with the rational investor's preference, that is, aiming for higher returns and lower variance. The rational investor will in fact prefer a portfolio with minimum variance for any specific level of return, which creates a curve along the feasible set of portfolios. The resulting parabola illustrated in Figure 2. was first introduced as the *critical line* by Markowitz (1959), and later became known as the *minimum-variance frontier*. Any portfolio not situated on this curve is regarded as inefficient as there exists a portfolio with the same return but less variance on the minimum-variance frontier. One portfolio in particular stands out, also known as the Minimum Variance Portfolio (MVP) as it has the lowest variance of all mean-variance portfolios.

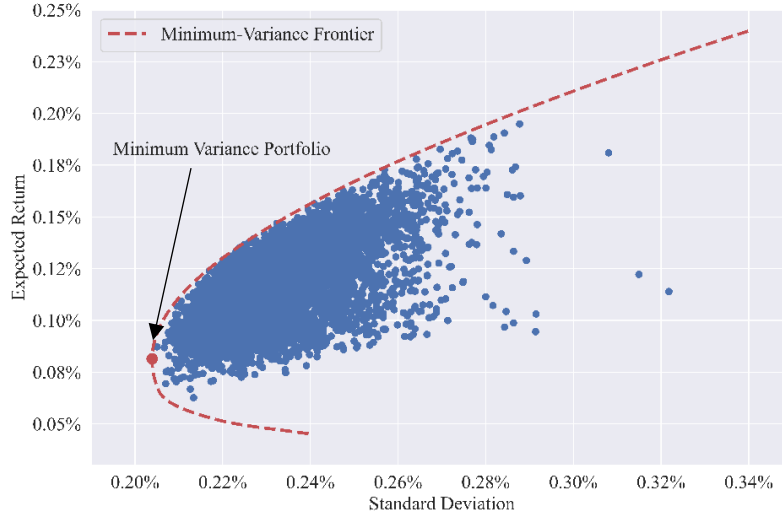


Figure 2. Illustration of the Markowitz Minimum-Variance Frontier with N risky assets. The figure has been created by the author.

II.1.3. The unconstrained optimisation problem

Having defined the framework of MPT, I will now move on to discuss the optimisation process. Suppose an investor who is equipped with data on N risky assets' expected returns (r_i , $\sum_i = N$) and the corresponding covariance matrix C , an $N \times N$ matrix containing the covariance between each two assets (the diagonal of the covariance matrix is each assets' covariance with itself, which denotes the variance). The investor seeks to find the optimal weights (w) for each security for any given level of return so that the created portfolio has the smallest possible variance. The resulting problem presented in Table 1. contains a quadratic objective function with linear constraints, hence it is a quadratic optimisation problem (Nocedal and Wright, 2006). When short-selling¹ is permitted, the problem at hand is also referred to as the unconstrained optimisation problem by Fabozzi *et al.* (2007). As the problem contains equality-constraints only, the optimal solution can be found with Lagrangian equations, however, in today's more complex cases, the process relies mostly on optimisation algorithms. Once the set of optimal weights outlining the minimum-variance frontier is found, the investment choice relies on the investor's preference for return and risk.

Objective	$\min_w w' C w$	minimise the portfolio variance
Subject to	$E(R_p) = w' r$	with given level of expected return
	$\sum w = 1$	the weights must sum up to 1

Table 1. The Markowitz Quadratic Optimisation problem with N risky assets

¹ The original framework builds on the prohibition of short selling however, Markowitz established the foundation for short sales to be permitted and there has also been a substantial amount of research on allowing it.

Ideally, the portfolios situated on the minimum-variance curve are not all desirable by the investor, as the rational investor is aiming to maximise their returns in addition to minimising the variance. When looking at the minimum-variance frontier on Figure 2, there are portfolios on the curve which are dominated by other portfolios, that is, portfolios with the same variance but higher return. The subset of the minimum-variance frontier in accordance with this condition is called the ‘efficient frontier’, which contains all minimum-variance portfolios with higher anticipated return than that of the MVP. The efficient frontier translates to first minimising the portfolio risk and then maximising the expected return. In Nocedal and Wright (2006), the two aims are combined and the joint optimisation problem in Table 2. is solved. In this case, the standard approach is to introduce a risk aversion parameter λ – also known as the “Arrow-Pratt risk aversion index” (Fabozzi *et al.*, 2007, p. 35) – to reflect the investor’s attitude towards risk. An investor with high appetite for risk will choose a lower value of λ and focus most of the optimisation on maximising the returns, whereas a rather risk-avoiding investor would appoint a high value to λ and align the decision mostly on minimising risk.

Objective	$\min_w \lambda w' C w - (1 - \lambda) w' r$	minimise the risk less return
Subject to	$\sum w = 1$	the weights must sum up to 1

Table 2. Optimisation along the efficient frontier

II.2. Improvements and current trends in the Markowitz model

Having introduced the original Markowitz framework, I will now turn to discussing the development of the model over the last 60 years. While the model itself meant a remarkable advancement to portfolio optimisation, it contained a number of limitations that numerous academics addressed in their research. These shortcomings are discussed in Zhang, Li and Guo (2018) and Guo (2022), outlining for example the lack of practical constraints, the model’s limitation to a single-period, the questionable robustness of the model and the choice of the decision criteria.

II.2.1. Dynamic portfolio optimisation

The most straightforward limitation to the mean-variance analysis was that Markowitz built his framework for single-period optimisation. As argued in several multiperiod optimisation frameworks, the investor’s choice on investment exceeds the assumptions of the single-period model. The optimisation should in fact be dynamic and take into account the investor’s present and future consumption as well as the growth of his income. One of the most significant improvements was introduced by Merton (1969) in his continuous-time model, where the

investor's choice of portfolio relied on intertemporal decision making. Merton introduced further uncertainty to his model by stating that the investor's income is based on stochastic returns. Fama (1970) suggested that the consumer's lifetime consumption is divided into multiple periods and the consumer should make their decision on consumption each period while considering the consumption in future periods. In the most recent multiperiod portfolio optimisation techniques, the mean-variance optimisation is leveraged for multiple periods rather than building a robust model that incorporates all time periods (Fabozzi *et al.*, 2007). The optimisation itself relies on minimising transaction costs associated with the multiperiod decision such as the cost of buying and selling assets during rebalancing. The portfolio is then reallocated at the beginning of each time period based on expected future returns over different scenarios.

II.2.3. Choice of decision criteria

Markowitz was the first academic who put portfolio optimisation and diversification in mathematical formulation through establishing expected returns and their standard deviation as decision inputs. However, his discovery did not imply that the mean-variance analysis became the single best method to optimising the portfolio and in fact laid down the foundation for future enhancements. Prigent's (2008, p. 82) comprehensive review concluded that the "*mean-variance analysis relies on "good" prediction of the mean and variance-covariance matrix of the securities*", explaining the need for a better estimation of the inputs.

The first significant contribution that followed the Markowitz framework was introduced by Sharpe (1963), who created the first index model. Apart from increasing the accuracy of the optimisation and forecasting, his work also simplified the original framework and the required computational power. As the single-index model became more popular, several researchers have explored the possibility of multi-index structures for estimating returns. The most prominent research in this direction has been carried out by Roll and Ross (1980) – the creators of the Arbitrage Pricing Theory –, as well as Fama and French (1992). Risk estimation also received considerable research attention, one particularly popular method was Value at Risk (VaR), which had been developed from the Markowitz model and was popularised in the 1990s by JP Morgan. More recently, Expected Shortfall (ES) is considered a better tool for measuring risk and promoting diversification Hull (2018).

II.2.4. Practical constraints

By far the majority of mean-variance analysis-related research has been concentrating on incorporating market imperfections within the framework through additional constraints. The aim of these studies has been to bring the Markowitz portfolio optimisation closer to the real-world problems. Such literature has been published by Fabozzi *et al.* (2007) and Prigent (2008), who both researched portfolio optimisation. The most straightforward constraint to implement in the optimisation is the prohibition of short selling which has already been addressed by Markowitz. When short selling is not allowed, the original problem is adjusted with the additional linear constraint $w \geq 0$. Transaction costs have also been the focus of several studies as they have a significant impact on returns and if not taken into consideration, they can diminish the expected returns that were used in the optimisation. One of the first approach to including transaction costs has been proposed by Pogue (1970), who made a comprehensive research on the inclusion of costs and fees related to portfolio construction. In practice, transaction costs act as another penalty function in the optimisation problem, adding to the effect of variance in the portfolio. The modified objective function is to minimise

$$\lambda w' C w + \lambda_{TC} TC - w' r$$

where TC refers to the penalty function and each source of loss is given a risk aversion parameter. Furthermore, there exist more complicated constraints on transaction costs that lead to the inclusion of nonlinear functions, which has been discussed by Yoshimoto (1996) and Kolm, Tütüncü and Fabozzi (2014).

Further constraints have been developed in order to limit the portfolio's exposure to particular industries, sectors or securities. In practice it is common to include lower and upper bounds on industries to ensure diversification in the portfolio. These linear constraints take the form of

$$L_i \leq w_i \leq U_i$$

where L_i ensures a minimum threshold for security i and U_i limits the maximum weight the portfolio can hold from the security. The holding constraints can be used for asset classes such as growth stocks and value stocks or small caps and big caps as well (Prigent, 2008). More complicated constraints include managing the portfolio relative to a benchmark or turnover constraints.

II.2.5. Cardinality constraints

With the emergence of numerous new forms of financial instruments such as green bonds and cryptocurrencies, the universe of tradable assets has expanded significantly since the Markowitz model was introduced, necessitating the limitation of the assets put in the portfolio. Cardinality constraints are a restriction on the number of assets that can be chosen in the portfolio Fabozzi *et al.* (2007). Many portfolio managers need to limit the total number of assets to satisfy certain investment objectives. This type of constraint is also quite useful when the portfolio is created against a broadly used benchmark such as the S&P 500 or the Russell 1000 that contain a large number of securities. In their comprehensive review, Zaimovic, Omanovic and Arnaut-Berilo (2021) concluded that while investment decisions depend on a wide range of factors, investors can diversify their portfolio with a relatively small number of assets (20-50 securities). Moreover, individual investors rarely hold large-sized portfolios as their transaction costs would exceed the benefits of holding such a portfolio. The general formula for cardinality constraints in the optimisation problem takes the form of

$$k \leq \sum_{i=1}^N z_i \leq K, \quad i = 1, \dots, N$$

where each z variable is a binary (0, 1) integer. In case of portfolio optimisation, $z = 1$ means that the particular asset is included in the portfolio, while $z = 0$ indicates that the asset was given 0 weight (that is, not included in the portfolio). Objectives that contain cardinality constraints are defined as integer programming problems. However, when the problem contains variables that are not limited to integer or binary values, the problem becomes a mixed integer programming (MIP) problem (Nocedal and Wright, 2006), which will be discussed in the following section.

II.3. Mixed integer programming

Mixed integer programming is a type of deterministic or mathematical optimisation method, which means that the optimisation is primarily relying on linear algebra. The major advantage of deterministic approaches is that through their rigorous mathematical formulation it is guaranteed that the solution found is unequivocal and definite (Cavazzuti, 2013). However, deterministic approaches search for stationary points in the response variable, therefore the solution reached may not be the global optimum.

According to the definition provided by Wolsey (2008), mixed integer programming refers to an optimisation problem where both integer and real-valued variables are present. These optimisation problems are typically solved by the branch and bound method, which is a stepwise iteration through the entire search space looking for the best solution. If both the objective function and the constraints are made up of linear equations or inequalities, the optimisation problem is a mixed integer linear programming (MILP) problem. However, as the Markowitz mean-variance analysis contains a quadratic objective function and linear constraints, the problem is a mixed integer quadratic programming (MIQP) problem (Lazimy, 1982).

Although mixed integer programming has drawn plenty of attention in optimisation problems, its utilisation in portfolio optimisation is limited in the research literature. The first attempt at applying the method to portfolio construction has been presented by Bertsimas, Darnell and Soucy (1999), who tested their MIP model on existing portfolios and found that their methodology was able to meet expected performance and reduce operational and transaction costs. Canakgoz and Beasley (2009) applied MIP to stock market index-tracking while implementing a number of constraints presented in section 2.2. The most notable contribution to the topic has been published by Mansini, Ogryczak and Speranza (2015), providing a significant analysis and discussion on the subject in their book ‘Linear and Mixed Integer Programming for Portfolio Optimisation’.

MIQP problems are much harder to solve than quadratic programming models – in the literature, they are often called NP² problems, referring to the complexity of the decision problem. Furthermore, in 2014 it has been proved that these problems classify as NP-complete problems (Pia, Dey and Molinaro, 2017). In Eiben and Smith (2015) and Garey, Johnson and Stockmeyer (1974), NP-complete problems are defined as problems that no algorithms can be relied upon to solve in a time that is polynomially limited by the input length. Fabozzi *et al.* (2007) interpreted this as the algorithms that are able to solve these problems are exponential in complexity, meaning that their runtime increases exponentially as the size of the problem grows. As such, these problems are computationally intractable and while there exist a number of solvers to optimise them, in practice they are rather solved with heuristics or other approximation methods.

² nondeterministic polynomial

III. Introduction to evolutionary computing

The next chapter provides an introduction on evolutionary computing (EC) to build a theoretical foundation for the primary focus of research in this paper. In order to better comprehend the benefits of applying evolutionary computing into finance and portfolio optimisation, this section will include an overview of the parameters required to develop evolutionary algorithms (EA), before outlining the structure and main components of the methodology. The primary source of literature used to create this chapter is the book ‘Introduction to Evolutionary Computing’ by Eiben and Smith (2015). The authors define evolutionary computing as an optimisation technique with metaphorical inspiration from the Darwinian evolution process.

Evolutionary algorithms fall under the category of stochastic or heuristic optimisation, which is mostly characterised by incorporating randomness in the search procedure. Their most notable advantage to deterministic optimisation is that through exploring the search space they are able to move away from the local optimum and find the global optimum if their runtime is not restricted. Stochastic optimisation techniques may not find the global optimum under limited time; however, they are able to find near optimal solutions.

III.1. Darwinian Evolution Theory

In order to help the reader better understand the concept and framework of evolutionary computing, a brief summary of the Darwinian Evolution Theory is necessary. The theory dates back to 1859 when Darwin published his book ‘The Origin of Species’ on evolution and natural selection. The core concept outlining the evolution process was the ‘survival of the fittest’, meaning that as the individuals of the population are competing for their environment’s finite resources the ones with better behaviour and physical abilities – concluded as phenotypic traits – will have a higher chance of reproduction.

The phenotypic traits measure the individuals’ capabilities to adapt to the features of their environment and translates to the fitness of the individuals. Part of these attributes are inherited while others are unique to each individual, partly as a result of random changes in their genetic code, also known as mutation. The difference in phenotypic traits creates a diverse population and is key to driving change in subsequent generations. However, the combinations of traits that have a higher chance of reproduction tend to be increasingly represented in the population. As a result of the natural selection and the survival of the fittest phenomenon, the fitness of the population will gradually increase. While the presence of highly fit individuals promotes higher

fitness of the population in consecutive generations, it is essential to note that the evolution process does not always go in an upward direction. In terms of optimisation, this can be translated as reaching a local optimum instead of finding the global optimum.

III.2. The four major Evolutionary Algorithm variants

The first attempts at utilising the Darwinian evolution theory in automated problem-solving date back to the late 1940s, its first appearance in literature is associated with Turing in 1948. In 1962, Bremermann was the first to examine evolution and recombination in the optimisation process. Over the next 30 years, four different evolutionary algorithm variants appeared and developed on their own, creating implementations of the same concept from various perspectives. The first realisation of an evolutionary algorithm was invented by Rechenberg and Schwefel in 1964, which became known as Evolution Strategies (ES) and was intended to solve difficult – discrete and continuous – parameter optimisation problems (Back, Hammel and Schwefel, 1997). At around the same time, Evolutionary Programming (EP) was created by Fogel, Owens and Walsh in the 1960s with the purpose of creating artificial intelligence. EP algorithms were originally designed to optimise a computer program that is encoded into each individual, however over the years they have become quite similar to ES (Dreżewski, Dziuban and Pająk, 2018).

The algorithms that have received most public and research attention are Genetic Algorithms (GA), which were introduced by Holland in 1975. In addition to their key advantage of being broadly adaptable to different kinds of problems, I chose to include GAs in the research methodology because they allow binary representation, which is quite suitable for portfolio optimisation. The last major branch of evolutionary algorithms is Genetic Programming (GP), which was created by Koza in the early 1990s and largely differs from the previous EA methods. A distinctive feature of GP algorithms is that they use a tree representation where each node denotes an algorithm, and the scope of the optimisation is to find the best possible algorithm for the given problem.

III.3. Main components of Evolutionary Algorithms

In this section, the main components forming evolutionary algorithms will be introduced. These components are irrespective of the evolutionary algorithm variant and in fact, the different variants can be distinguished by the differences in their components. The components will be presented in the order they appear in the optimisation process displayed on Figure 3. It is also

a common practice to describe any algorithms in pseudo code format, – an informal representation of the algorithm describing its computing process – which will also be used to demonstrate the genetic algorithm formulated in this paper.

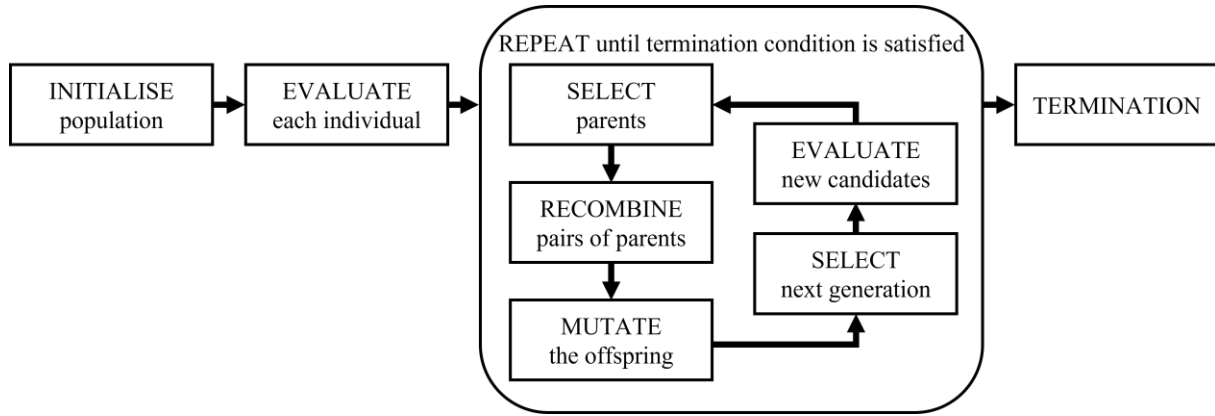


Figure 3. Evolutionary Algorithm flowchart. The figure has been created by the author.

III.3.1. Representation

Evolutionary algorithms are modelled from the real-world evolution process, therefore the first step of an EA is to define how the elements of the population will be mapped to the EA framework. In genetics, phenotype represents the observable traits of an individual, which is determined by the genotype. The phenotypic information is encoded in the genotype in the form of chromosomes, or strands of DNA. Therefore, the candidate solutions forming the problem are denoted as the phenotype, while their EA representation is the genotype – a set of chromosomes that explicitly determine each individual. During the initialisation of the EA algorithm, the phenotypes of randomly generated candidate solutions are encoded into genotypes. The representation of candidate solutions must be invertible as the solution of the algorithm must be decoded from genotypes to phenotypes. The EA can only find the global optimum if every feasible solution is encoded into computationally tractable genotypes, which is often called the genotype space.

III.3.2. Population

As a result of the initialisation process, all candidate solutions of the problem are represented by their genotype. The population represents the entirety of possible solutions, which is a multiset of genotypes as repetitions of the same phenotype encoding is possible. The population acts as the basic unit of evolution, which means that the population is evolving, not the individuals. This also implies that the parents and survivors are selected on population level. In classical cases, the population can be defined by its size, however, in more sophisticated EAs

the spatial structure – that is, the distance between the individuals – can also hold useful information about the population. During the evolution process, the population size is usually constant, indicating the limited resources the individuals need to compete for. The diversity of the population refers to the number of different solutions present. While diversity can be measured by different methods, it is a common practice to determine it by the number of different genotypes in the population. Diversity is an important feature of the population as it is accountable for preventing the population from shifting to a local optimum.

III.3.3. Evaluation function

Once the initialisation process is finished, the next step for the EA is to evaluate each individual. In terms of optimisation, the evaluation also corresponds to the objective function, that is, the requirements the candidate solution has to meet. The direct connection to the evolution process comes from the fact that in any specific environment, some phenotypic traits grant more advantages such as thicker skin provides more protection from predators or other environmental factors. During the evaluation process, desirable traits will be rewarded by the evaluation function while unwanted features will be penalised, providing a basis for comparison between individuals. The evaluation function assigns a fitness score to each individual hence the function is also called a fitness function. In many optimisation problems the fitness is being maximised, however, minimisation is also common and both versions can be converted into the other. In this analysis, the presented algorithm contains a fitness function with minimisation as its scope.

III.3.4. Selection mechanism

The selection mechanism can occur in two stages and is designed to increase the average fitness of the solutions in the population (if the problem is a maximisation problem). At first, the selection mechanism determines the parents for offspring creation. The input to the selection mechanism is the fitness score, meaning that the individuals with better fitness score have a higher chance to survive the selection process. However, in order to avoid getting trapped in a local optimum, the selection mechanism is usually probabilistic. This means that while high-quality solutions will have a higher probability of being selected, even the individual with the worst fitness in the population has a nonzero probability of being selected as parent.

Once the offspring is created and has been through mutation, the selection mechanism identifies the individuals to survive and become part of the next generation. In order to comply with the ‘survival of the fittest’ principle, contrary to parent selection, the survivor selection is rather deterministic. Therefore, the survivor selection targets individuals with low fitness and replaces

them with the generated offspring. The two most common methods of choosing members to be replaced are 1) the candidate with the worst fitness or 2) the first member with a fitness lower than the given offspring.

III.3.5. Variation operators

Variation operators are responsible for creating the necessary diversity within the population by generating new candidate solutions from old ones. Depending on their arity (number of inputs), variation operators can be divided into two broad categories. A variation operator with an arity of 1 is called ‘mutation operator’ and its role is causing small, random variances in the population. In the EA process the mutation is usually happening after the recombination (if the algorithm contains recombination), thus implying that the offspring is the subject of the mutation. The mutation is a stochastic process – the changes in the subject’s genotype are always random and can result in both higher and lower fitness of the new genotype. The importance of the randomness is to guarantee that each genotype representing a possible solution can be reached by the variation operators, hence avoiding convergence to suboptimal solutions.

Variation operators with an arity higher than 1 are known as recombination or crossover operators. While biologically this implies that the arity equals two, in some rare cases of EC more than two parents are used in creating new candidate solutions, since the process is mathematically possible. The recombination operators are used to merge the genotype information from the selected parents into one or two offspring. Similarly to mutation, the crossover is also stochastic, meaning that the choice of what characteristics of each parent are combined is random. Since the parent selection is targeting individuals with more desirable traits, the role of recombination is to create a child that holds good traits from both parents, thus increasing the fitness of the offspring. The recombination and crossover are usually both present in evolutionary algorithms and their combined use is able to improve the fitness value in subsequent generations.

III.3.6. Termination

Starting from the initial population, the evolutionary algorithm runs through many generations before fulfilling its termination criteria. In each generation, parents are selected to create the offspring, which go through a mutation, before replacing individuals with worse fitness. In an ideal setting, the EA runs until the optimal solution is found, which can either be a known optimal fitness level or a satisfactory approximation of a desired fitness. However, because of

the stochastic nature of evolutionary algorithms, the realisation of this optima is not guaranteed. Instead, the termination depends on the fulfilment of a condition that is pre-defined by the user. The first option is to allow the EA to run for a specified amount of time, and once that is reached the algorithm will terminate. It is also common to establish a given number of generations after which the EA must stop. The third option is to terminate the algorithm if the population drops under a certain level of diversity. Finally, it is also recommended to terminate the algorithm if the population reaches a specified number of generations without material fitness improvement. While the algorithm is running, it will evaluate the termination condition for each generation, and will stop once this condition is satisfied.

III.4. Genetic Algorithms

The following part of this paper moves on to describe genetic algorithms in greater detail as this EA variant will be implemented in the research. I will first entail how each of the components are featured in GAs, then discuss the typical application areas and most common use cases. The section will also include simple examples on the workflow of genetic algorithms to build the necessary foundation for the methodology presented in chapter 4. Lastly, I will move on to discuss the advantages and disadvantages of GAs, as well as the improvements made to the original algorithm over time.

III.4.1. The structure of genetic algorithms

Genetic algorithms are the most widely used type of evolutionary algorithms, they are also considered as the classical or simple genetic algorithm (SGA). Because of their simple and transparent structure, they often serve as the basis of teaching EAs, and they can also be used for benchmarking purposes for building new algorithms. The representation of GAs is bit-strings, meaning that the genotypes consist of binary values, such as the following encoding

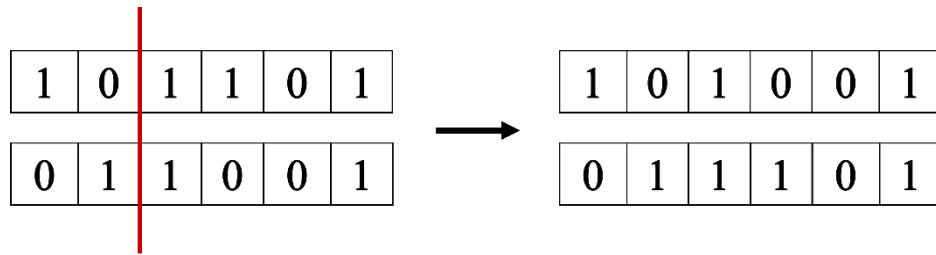
1	0	1	1	0	1
---	---	---	---	---	---

where each gene represents a particular characteristic of the individual. If a decimal value was to be represented by binary numbers, the above example would be decoded to

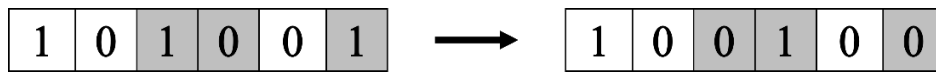
$$1 * 32 + 0 * 16 + 1 * 8 + 1 * 4 + 0 * 2 + 1 * 1 = 45.$$

The reproduction cycle of SGAs consists of 5 steps. At first, the parents are selected for the mating pool where traditionally the size of the mating pool is the population size while duplicates are allowed. The parent selection is fitness proportional, meaning that the probability

of each individual to be chosen as parent equals its fitness in proportion to the fitness of the whole population. The mating pool is then shuffled, and each parent is paired randomly with another to undergo the recombination. The recombination operator in case of genetic algorithms is usually the 1-point crossover, which essentially means that a random crossover point is selected along the bit-strings and denotes the point after which the genes of the parents are swapped, creating two offspring:



Following the recombination, the mutation is carried out through bit-flipping with a given probability for each offspring. In practice, this mutation operator runs bitwise flipping along each gene, allowing it to flip (from 0 to 1 or from 1 to 0) with the same probability. If the probability takes the value of 0.5 it is possible that the mutation will go as follows:



However, as the probability of mutation is independent for each bit, it is possible that more or less genes will be mutated than exactly half of them. The last step of the reproduction cycle is replacing the whole population with the created offspring, thus replacing the entire population with a new generation. The survival selection is therefore called ‘generational’.

III.4.2. Improvements to genetic algorithms

Genetic Algorithms were first introduced by Holland (1975) as a way of studying adaptive behaviour. While the original SGA framework contained the important foundations that are still used today, it required too much runtime and was missing important variants such as elitism and stochastic universal sampling (SUS), that have been added to the model later. Elitism is a method that helps evolutionary algorithms to reach faster convergence and is widely used in EAs. The concept of elitism is to sustain the currently fittest member of the population by only replacing it if the offspring has equal or better fitness. Stochastic universal sampling is an enhanced selection mechanism proposed by Baker (1987) that allows for selecting parents with better fitness.

Today genetic algorithms are widely used in different fields such as economics, machine learning, optimisation, engineering and politics (Katoch, Chauhan and Kumar, 2021). While new streams of alternative GAs such as Linkage Learning, multi-objective GAs and Messy genetic algorithms have been developed since the late 1980s, simple GAs are still quite useful for benchmarking and for relatively straightforward problems that allow binary representation.

III.5. Advantages and disadvantages of evolutionary algorithms

Evolutionary algorithms became popular very quickly amongst optimisation algorithms and opened the door to many other stochastic optimisation algorithms mimicking processes in the nature. Together with other stochastic optimisation variants, EAs are quite effective in exploring the search space, making it possible to reach solutions that are inaccessible by mathematical methods. EAs are therefore able to reach the global optimum instead of the local optima, although they usually require significant amount of time to achieve that. The randomness in their processes also allows them to find solutions more effectively in discontinuous or non-differentiable search spaces (Schwefel, 2000), making them easily adaptable in various fields of problem areas.

Unlike deterministic algorithms, EAs are also able to handle multiple objectives, which received a lot of research attention in the last 30 years. Optimising for multiple objectives is also known as Pareto optimisation which appears in many practical problems where two or more objective functions are to be solved simultaneously, typically including trade-offs. EAs have become quite efficient in solving these problems with dividing the population into subpopulations corresponding to each objective (Schaffer, 1985) or establishing a dominance between the objectives through the fitness function (Goldberg, 1989).

The third main advantage of EAs is their flexibility to adapt to a wide range of problems. They can be used in many different areas where conventional methods are less effective such as in dynamic optimisation, where the objective or the constraints are changing in time. The robustness of EAs also allows them to gain advantage over mathematical algorithms in situations where noise or uncertainty is involved. Through the broad variety of parameters, they are also easily adjustable to the user's preference for example in finding better solutions or reaching faster convergence.

However, as implied by the no-free-lunch theorem, EAs are not superior to any other methods and their effectiveness in some areas comes at a price of worse behaviour in other problems. For instance, while EAs are able to overcome the local optimum and find the globally best

solution, as a consequence they have a much slower convergence to the global optima. They are also less effective at solving simpler problems such as linear optimisation and as suggested by Schwefel (2000), they should not be used in areas where traditional solutions exist. The main cause of this is that EAs could not solve these problems more effectively or with less computational effort. Finally, while in some circumstances the presence of randomness in the EA processes can be quite beneficial, if the user is seeking to find more accurate solutions, deterministic algorithms can reach these more easily and with more transparency.

IV. Methodology

This chapter provides insight into the procedures and methods implemented in this analysis to test the effectiveness of genetic algorithms. In order to evaluate the performance of evolutionary computing, in particular genetic algorithms, a functional GA was built to solve the cardinality constrained optimisation problem. The performance indicators of the GA are compared to those of an MIP algorithm, which represents the typical approach to solving cardinality constrained problems. Additionally, both methods are investigated in relation to the unconstrained problem as a means to examine their deviation from the global optimum. The algorithms are tested in a simulated environment with randomly generated returns and variance; however, section 4.5. will also provide a framework for testing their performance on real data.

The problem to be solved by each algorithm is to optimise a portfolio by choosing a specified number of assets from a broader investment universe. The algorithms are expected to choose the correct number of assets and assign them the weights the portfolio should hold from them so that its mean-variance positioning is optimal. In particular, the algorithms will be tested with increasing problem size as traditional mixed integer programming requires much computational power in a large-scale optimisation problem. While other constraints were also discussed in section 2.2.4., those constraints are not in scope of the optimisation algorithms presented in this paper as their implementation is computationally expensive and requires much greater complexity. That said, incorporating cardinality constraints builds the necessary amount of complexity in the optimisation problem and is also a widely recognised method in portfolio optimisation by Chang *et al.* (2000) and Fieldsend, Matatko and Peng (2004), among others.

IV.1. The unconstrained problem

Before demonstrating the genetic algorithm as the primary scope of this analysis, it is essential to first illustrate the algorithm for the unconstrained optimisation problem. The unconstrained method will not only serve as a benchmark for performance comparison but will also be used as the genetic algorithm's evaluation function. A pseudocode explaining the main steps of the quadratic optimisation is displayed in Figure 4. The algorithm is aiming to optimise the traditional Markowitz mean-variance optimisation problem where short selling is prohibited and the algorithm distributes the weights optimally between all assets. The inputs to the quadratic optimisation are the number of assets (N), the expected returns (r_i), the covariance matrix (C_{ij}) and the risk aversion parameter (λ). The objective is to minimise the quadratic risk minus return formula on row 3 so that the weights are non-negative and add up to 1.

```

optimise( $N, \lambda, r_i, C_{ij}$ )
// Let  $N$  denote the problem size
// Let  $\lambda$  denote the risk aversion parameter
// Let  $r_i$  denote the expected returns,  $i \in N$ 
// Let  $C_{ij}$  denote the covariance matrix
// Let  $w_i$  denote the weight assigned to each asset,  $i \in N$ 
// Let  $f$  denote the optimal objective

BEGIN
01  $\sigma_P^2 = \sum_i \sum_j w_i w_j C_{ij}$ 
02  $r_P = \sum_i w_i r_i$ 

// Optimisation
03 min  $\{f = \lambda \sigma_P^2 - (1 - \lambda) r_P\}$ 
04 subject to  $\sum_i w_i = 1$ 
05  $w_i \geq 0$ 

// Return optimal objective and weights
06 return  $f, w_i$ 
END

```

Figure 4. Algorithm: Unconstrained optimisation. The figure has been created by the author

The quadratic programming problem is solved using the Gurobi Optimization software, a frequently used solver for both quadratic and mixed integer programming. While there is no single best software for this purpose, Gurobi is widely recognised for its fast performance and suitability for a broad variety of problem domains. The software is also adaptable to the Python programming language which was used to build the algorithms for this paper.

IV.2. The genetic algorithm

The major components of evolutionary algorithms have been outlined in section 3.3., however, implementation methods greatly differ across evolutionary algorithms. Therefore, the algorithm structure will be outlined and discussed for each component in the following. The genetic algorithm has been designed to effectively solve the portfolio optimisation problem in the presence of cardinality constraints. The algorithm that is developed and tested in this paper is summarised in a pseudocode in Figure 5, while the full version along with its components is included in Appendix 1. The inputs for the algorithm are: the universe of investable assets (N); the number of instruments that can be selected (K), the vector of expected returns (r_i) and the covariance matrix (C_{ij}).

```

// Let  $N$  denote the problem size
// Let  $K$  denote the cardinality constraint
// Let  $P$  denote the population size
// Let  $S_p$  denote the candidate solution,  $p \in P$ 
// Let  $T$  denote the wall-clock time
// Let  $r_i$  denote the expected returns,  $i \in N$ 
// Let  $C_{ij}$  denote the covariance matrix
// Let  $w_i$  denote that weight assigned to each asset,  $i \in N$ 

BEGIN
// Initialisation
01 generate  $P$  population of  $N$  random individuals  $S$  in  $(0, 1)$ 

// Evaluation
02  $\sigma_P^2 := \sum_i \sum_j w_i w_j C_{ij}$  for all  $S_{i,j \in N} = 1$ 
03  $r_P := \sum_i w_i r_i$  for all  $S_{i \in N} = 1$ 
04  $f, w_i = \text{optimise}(K, \lambda, r_P, \sigma_P^2)$ 

// Reproduction cycle
05 while  $T < \text{time limit}$  do
    // Parent selection
    06 select  $S_1, S_2 \in P$  by binary tournament
    // Crossover
    07 set random  $p_c := \{p_1, \dots, p_N\}$  in  $[0, 1]$ 
    08 if  $p_i < \text{crossover rate}$  then  $S^*[i] = S_1[i]$  else  $S^*[i] = S_2[i]$ 
    // Mutation
    09 if  $\sum_{i \in N} S_i^* > K$  then
        10 while  $\sum_{i \in N} S_i^* > K$  do randomly replace a 1 with 0 in  $S^*$ 
    11 if  $\sum_{i \in N} S_i^* < K$  then
        12 while  $\sum_{i \in N} S_i^* < K$  do randomly replace a 0 with 1 in  $S^*$ 
    // Evaluate offspring
    13  $\sigma_P^2 := \sum_i \sum_j w_i w_j C_{ij}$  for all  $S_{i,j \in N} = 1$ 
    14  $r_P := \sum_i w_i r_i$  for all  $S_{i \in N} = 1$ 
    15  $f, w_i = \text{optimise}(K, \lambda, r_P, \sigma_P^2)$ 
    // Select next generation
    16  $P[\min \{f\}] = S^*$ 
    17 end while

18 return  $S, f_S, w_S$ 
END

```

Figure 5. Algorithm: Genetic Algorithm. The figure has been created by the author.

IV.2.1. Representation and initialisation

The representation follows the traditional GA principles, therefore the candidate solutions are represented in a bit-string format, where 1 indicates that the particular asset is chosen in the portfolio and 0 means that the asset is not included. Each phenotype represents a portfolio of

the chosen assets which are encoded into the binary genotype where the order of the numbers does not hold any additional information. During the initialisation, P individuals are created randomly, their chromosome contains N genes each. The fundamental rule for the initialisation is that each individual can contain only K ones (hence $N - K$ zeros), where $K < N$. In this paper, the population size was set to 100, a frequently used size presented in both Eiben and Smith (2015) and Chang *et al.* (2000).

As part of the simulation, the expected returns and the variance are also generated randomly. The expected returns are stored in a vector of N length and are generated with a normal distribution. The variance takes the form of a covariance matrix and is also generated randomly while ensuring that the feasibility conditions for quadratic programming are also fulfilled, that is, the matrix is either positive definite or positive semidefinite.

IV.2.2. Evaluation function

The evaluation function is giving each candidate solution a measure of quality based on the Markowitz mean-variance analysis. The fitness of each individual derives from the optimal value of the objective function

$$\min \lambda \sigma_p^2 - (1 - \lambda)r_p,$$

which is optimised for the unconstrained optimisation problem containing the selected assets. This means that the quadratic programming is only performed on those assets whose genes have the value of 1, and accordingly, the return vector and covariance matrix are also filtered for these assets. The benefit of such an evaluation function is that, since the fitness value can only be obtained by solving the unconstrained optimisation problem, running the algorithm does not only optimise which assets get to be chosen in the portfolio but also gives them the appropriate weights. As the objective function contains minimisation, the goal of the genetic algorithm is to minimise the average fitness of the population in subsequent generations. Therefore, species that contain assets with higher returns and lower variance are more desirable throughout the evolution process. The value of λ is chosen to be 0.75 when running the algorithm, representing a rather risk-avoiding investor.

IV.2.3. Parent selection

The choice of parent selection method in this genetic algorithm is binary tournament selection, which chooses two parents to take part in recombination. The term ‘tournament selection’ means that the selection mechanism is picking individuals from a mating pool based on their

fitness. To ensure that each individual has a nonzero chance of getting in the mating pool, the pooling is done at random, with a tournament size of 2, – hence, binary tournament selection – meaning that both pools consist of two individuals. The selection mechanism then picks the candidate with lower fitness score from each pool to become parents as a means of favouring individuals with better traits to carry on the evolution process. The major advantage of tournament selection is that the mechanism decides based on relative rather than absolute fitness, preventing the formation of bottlenecks at this stage. The procedure is highly useful for bigger populations, which is the case for the genetic algorithm in this analysis.

IV.2.4. Recombination and mutation

The recombination and mutation are described in the pseudocode (Figure 5.) through lines 7-12. The selected variation operator for recombining parents is called uniform crossover, which builds up the chromosome-string by treating each gene independently and choosing it randomly from one of the parents to inherit. The important tuning parameter of the uniform crossover is the crossover rate, which determines how much of the genes are to be transmitted from each parent. When the crossover rate of one parent is p , the crossover rate of the other is $1 - p$.

During the crossover, a string of random variables in $[0, 1]$ is created from a uniform distribution, where the length of the string equals the number of genes in the parents – thus the number of crossover variables created in this GA is N . The mechanism then goes through each gene in the offspring and if the random variable is higher than the crossover rate, the gene will be inherited from the first parent, and from the second if it is lower, such as in the following example in Figure 6. This way if an asset is present in both parents, it will be passed down to the child as well, and in case only one parent possesses the given asset, then the child will have a p probability of inheriting it.

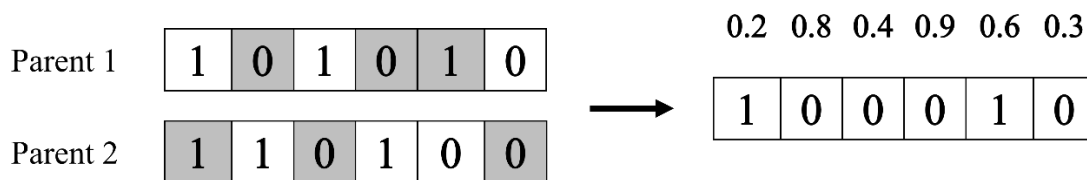


Figure 6. Illustration of uniform crossover, where crossover rate = 0.5

The example illustrates the uniform crossover with a crossover rate of 0.5 that has been chosen for the GA in this paper, where both parents have equal chance of transmitting their genes to the offspring. In other GAs it is also common to favour one of the parents by assigning a higher

crossover rate to it, however, since the two mating pools in this algorithm are independent from each other, the crossover is not biased to any of the parents.

The mutation process in this GA serves two purposes: the first is to introduce diversity to the evolution process, and the second is to ensure that the number of selected assets remains constant in consecutive generations. A direct consequence of any recombination operators in case of binary representation is that the resulting offspring might have more or less ones in its chromosome than the parents, since the inheritance of genes is done randomly. An example of this phenomenon is demonstrated in Figure 6. Even though both parents have the same amount of zeros and ones in their chromosome, the crossover does not guarantee that the offspring will do so as well. Therefore, in order to keep the number of chosen assets at constant, some kind of mutation is inevitable in this genetic algorithm. This way, while the mutation operator fulfils its primary purpose of exploiting new solutions and facilitating changes in the population, its secondary purpose is to restore the number of chosen assets to K after the recombination. If the crossover results in the sum of ones higher than K , the mutation operator randomly flips a 1 to 0 until the offspring has the same number of ones as the parents; and vice versa, if not enough assets are selected, the mechanism will flip a 0 to 1 randomly as many times as needed. In the rare cases when the offspring has the same ones and zeros as the parents, the mutation does not take place.

IV.2.5. Survivor selection and termination

The survivor selection is done on the basis of worst fitness, meaning that the member with the weakest fitness value will be replaced by the offspring in each generation. While this method can lead to convergence to a suboptimal solution, it helps to improve the average population fitness quickly. With larger population size the possibility of premature convergence can be lowered, whereas replacing only one individual also helps to prevent this issue. The delete-worst method also corresponds with the concept of elitism as the member with the current best fitness will always stay in the population, thus improving the convergence to the globally optimum solution.

The termination condition is set to a given wall-clock time T , meaning that the GA stops once the allowed wall-clock time has expired and the solution is the individual with the current best (lowest) fitness score. The decision was primarily driven by the motive to enable comparison with the MIP algorithm. While the approximation of a global optimum could also be considered, the runtime of both the GA and the MIP would grow exponentially as the problem size increases

(Ouaïl and Chergui, 2018). Therefore, the other reason for putting an upper bound on time for both algorithms was to mitigate their excessive time consumption. This creates a trade-off between optimality and feasibility, a controversy that is commonly featured in approximation algorithms (Hochba, 1997).

Wall-clock time can be defined as the time that has elapsed from the start until the end of the process that the user could measure on a stopwatch. It vastly differs from the other commonly used time measure, the CPU time, which is the amount of time the CPU has spent processing instructions and data for a specific program (Roussel, 2011). CPU time does not include the time the processor spent sitting idle or on executing other programs. However, modern processors are so advanced that they usually support two or more cores, that allows them to run processes on multiple threads (Eiben and Smith, 2012). In case of such complex processes like evaluating fitness for evolutionary algorithms, the CPU time is made up of the time the processor spent on each thread, increasing the total CPU time substantially (Scott and De Jong, 2015). As an example, a processor that features four cores has four CPU seconds for every wall-clock second. Therefore, while CPU time can be useful for measuring performance, wall-clock time is more suitable as the termination condition.

IV.3. The MIQP algorithm

This section establishes the framework for the MIQP algorithm, which is designed to solve the same cardinality constrained optimisation problem utilising the Gurobi solver. The main steps of the optimisation process are summarised in the pseudocode in Figure 6. The MIQP algorithm is built on the foundation of the unconstrained problem, but with the addition of the cardinality constraint throughout rows 8-9. The limit on the number of assets is characterised with a vector of indicator variables, which are binary numbers indicating whether the asset is included in the portfolio or not. The upper bound is imposed through a variable called big-M (MOSEK ApS, 2022), which is typically set at a large number such as 1000. Since the weights are required to be non-negative, a positive weight implies that the indicator variable is 1, whereas $w = 0$ can only be possible if the indicator variable is 0. The other element of the constraint requires the sum of the binary variables to be equal to the number of assets chosen in the portfolio, ensuring that the problem is optimised for the correct number of assets and the optimal weights.

Just as the GA, The MIQP algorithm is also subject to a time limit T , which prompts the algorithm to return the current best solution found under the limited amount of wall-clock time. As an NP-complete problem, running an MIQP solver without time constraints would be

impractical in a large-scale optimisation framework as the additional time consumption is unlikely to substantially increase the improvement in the optimal solution.

```

optimise( $N, \lambda, r_i, C_{ij}$ )
// Let  $N$  denote the problem size
// Let  $K$  denote the cardinality constraint
// Let  $\lambda$  denote the risk aversion parameter
// Let  $T$  denote the wall-clock time
// Let  $z_i$  denote a binary indicator variable,  $i \in N$ 
// Let  $r_i$  denote the expected returns,  $i \in N$ 
// Let  $C_{ij}$  denote the covariance matrix
// Let  $w_i$  denote the weight assigned to each asset,  $i \in N$ 
// Let  $f$  denote the optimal objective

BEGIN
01  $\sigma_P^2 = \sum_i \sum_j w_i w_j C_{ij}$ 
02  $r_P = \sum_i w_i r_i$ 

// Optimisation
03  $M = 1000$  fixed upper bound
04 while  $T < \text{time limit}$  do
05   min  $\{f = \lambda \sigma_P^2 - (1 - \lambda) r_P\}$ 
06   subject to  $\sum_i w_i = 1$ 
07              $w_i \geq 0$ 
08              $w_i \leq z_i * M$ 
09              $\sum_i z_i = K$ 

// Return optimal objective, chosen assets and weights
10 return  $f, w_i, z_i$ 
END

```

Figure 7. Algorithm: Mixed Integer Quadratic Programming. The figure has been created by the author.

IV.4. Comparison of the two methods

This section presents the methodology for comparing the performance of genetic algorithms and mixed integer quadratic programming in the constrained portfolio optimisation problem. GAs and MIQP represent greatly different optimisation models, the major difference being the stochastic versus deterministic optimisation approach. Their comparison in portfolio optimisation is not commonly featured in research literature; therefore this study intends to show empirical results on applying both methods to different forms of the same problem.

In related literature, Costa and Oliveira (2001) studied both GAs and ESs for solving mixed integer nonlinear problems, whereas Rothberg (2007) considered EAs as an improvement to

MIP models and used EAs incorporated into the MIP framework. Nemati, Braun and Tenbohlen (2018) applied GAs and MILP to energy management optimisation and showed that MILP yielded better results with few constraints, whereas in a highly constrained environment GA found better solutions. Kuendee and Janjarassuk (2018) compared MILP and GAs for binary problems and came to the conclusion that GAs worked faster than the MILP method, however, MILP provided solutions of higher quality. Finally, Peters *et al.* (2019) compared MIP solver Gurobi and EAs based on their speed to solve a staff assignment optimisation problem and found that their EA implementation was able to find a near optimal solution in less time than Gurobi.

IV.4.1. Problem setting

The two algorithms presented in sections 4.3. and 4.4. are tested in the same environment with the primary focus of research being the best objective that is found in the presence of time boundary. The optimal solutions of both approaches are compared to the optimum found by the unconstrained solver. The primary motive for this is to assess the deviation between the situation in which the entire asset universe is included and when only select assets are held in the portfolio.

The investment universe ranges from $N = [500, 5000]$ with intervals of 500, whereas the wall-clock time limit is set at 10, 15, 30 and 60 seconds in each execution. To adjust the number of selected assets to the problem size, the chosen assets also differ between $K = \{15, 100, 250\}$. Each combination of parameters corresponds to a different problem setting, one example of which is $N, K, T = \{1000, 100, 30\}$. The simulation iterates through each of these settings with the first level being the investment universe. For each different value of N , a new return vector and covariance matrix are created with the corresponding size and the three algorithms are required to optimise for the different values of K under each time constraint. There are a total of 10 simulations for each investment universe to create the necessary sample size for hypothesis testing, therefore the entire analysis consists of 1200 instances for each algorithms.

Through the different combinations of parameters, the experiment was performed in a total of approximately 10 hours for both methods. This reduces the time and computational power the algorithms consume and is also broadly applicable to other practical optimisation situations where the solver has to meet a deadline or there is a high cost associated with a long execution time. In particular, the time budget prevents the formulation of bottlenecks in case of larger investment universes while still arriving at a good solution.

IV.4.2. Constructing hypotheses

The primary expectation of the analysis is that compared to the MIP solver, genetic algorithms are able to find higher quality solutions with increasing problem sizes. As concluded from previous research literature, GAs are more likely to avoid local optimums and are expected to gain advantage from the fact that they can explore the search space through variation operators. In order to evaluate these expectations, two separate hypotheses were constructed.

- 1) The first, right-tailed hypothesis assesses the quality of the optimal solution in a small asset universe:

H_0 : GAs will reach lower optimal objective than the MIP solver.

H_1 : GAs will reach higher optimal objective than the MIP solver.

- 2) The second, left-tailed hypothesis is constructed for the large investment universe:

H_0 : GAs will reach higher optimal objective than the MIP solver.

H_1 : GAs will reach lower optimal objective than the MIP solver.

IV.4.3. Statistical tests

To determine the validity of the hypotheses, a significance test will be run to compare the means of the optimal solution for the two methods. In this case, two main types of statistical tests can be used based on the assumptions made of the sample: the dependent samples t-test and the Wilcoxon signed rank test. The dependent samples t-test (Student, 1908) is a parametric statistical test that compares two samples from normally distributed populations. Since this test requires an assumption about the normality of the two samples, before conducting the t-test, a preliminary simulation was carried out to test the normality of these samples. Using the Shapiro-Wilk test for normality, each problem setting was tested to decide whether the means of the optimal objective values belonged to a normal distribution for both methods. While in most cases the null-hypothesis could be accepted, not all samples belonged to a normal distribution, therefore the t-test has been discarded.

In order to maintain consistency, the Wilcoxon signed rank test (Wilcoxon, 1945) was chosen for hypothesis testing since it does not require the samples to approximate normal distribution. This test is the non-parametric version of the dependent samples t-test which ranks the differences between the sample means and assigns negative signs to each negative difference. The differences are considered insignificant when their mean signed rank is 0, otherwise the

difference is statistically significant. The Wilcoxon test was carried out for each problem setting with a sample size of 10. The significance level was chosen to be 95%, implying that the probability of the null hypotheses being falsely accepted is 5%.

IV.5. Testing with real-world data

While the research is primarily carried out in a simulated environment for the purpose of generalising results, a small segment of the analysis is dedicated to testing the algorithms in a real data environment. The data set comprises of daily prices ranging between 1 January 2018 and 31 December 2022 and was exported from Yahoo Finance for 5200 stocks³ listed on the New York Stock Exchange.

IV.5.1. Problems with real data usage

Both the GA and the MIP in this paper are created to be highly adaptable to optimising portfolios in any testing environment as the only required inputs are the expected returns and the covariance matrix. Running the optimisation for a real data set might be beneficial as a means of removing the assumption that the asset returns follow a normal distribution. However, optimising for a sample of real-world data also has several drawbacks as presented by (Jobson and Korkie, 1980) and (Ledoit and Wolf, 2004).

The main challenge is raised by the estimation of the covariance matrix from a sample of asset returns. As shown by (Ledoit and Wolf, 2004), if the covariance matrix is estimated for a large number of assets, it tends to be calculated with a large estimation error. This phenomenon is further amplified when the number of available historical returns is relatively small, which is typically true. This causes the quadratic optimisation function to falsely assign high weights to a number of assets with highly inaccurate values in the covariance matrix. Ledoit and Wolf proposed a shrinkage estimator that is able to reduce extreme values in the covariance matrix.

When the number of return observations is smaller than the universe of assets, it is also common that the estimated covariance matrix is not positive definite, which implies that the solver may not find an optimal solution for the quadratic optimisation problem. Further problems may arise when there is a high correlation or linear dependency between asset returns, which is usually the case for different share classes of the same stocks. This causes the covariance matrix to become singular and thus, not positive definite.

³ The number of stocks for which daily price data was available for the whole 5-year period.

IV.5.2. Calibration for real data testing

The Yahoo data set of daily prices was used to calculate mean returns for the given time frame, while the covariance matrix was calculated in Python from the daily returns over the same 5-year period. To overcome the estimation error of the covariance matrix and ensure that the matrix is positive definite, the original covariance matrix was modified using the Ledoit and Wolf covariance shrinkage formula with Python's built in method⁴. The formula shrinks the sample covariance matrix to a more accurate version using the following formula

$$\delta F + (1 - \delta)S$$

where F is the shrinkage target denoting an ideally structured covariance matrix⁵ and δ stands for the shrinkage constant, which is a weighting parameter between the optimal covariance matrix F and the sample covariance matrix S . The shrinkage method is able to bring the variances closer to each other by adjusting the outliers while keeping the diagonals intact.

The modified covariance matrix and the mean returns are used as input to the GA and the MIP solver, calibrated with the same parameters as follows below:

investment universe, N	[500, 5000], 500 increments
cardinality constraint, K	{15, 100, 250}
wall-clock time limit, T	{10, 15, 30, 60}
risk-aversion parameter, λ	0.75
population size (GA only), P	100
tournament size (GA only)	2
crossover rate (GA only)	0.5

The results of running the algorithms in real data environment are tested with the same hypotheses and the Wilcoxon signed rank test to evaluate the differences between the optimal objectives in the presence of a time constraint.

⁴ LedoitWolf class in Python's sklearn module

⁵ In the Ledoit and Wolf framework this translates to a covariance matrix where all variances have equal values and the covariances are zero.

V. Discussion of empirical results

The results of the analysis are presented in this chapter for both the simulated and the real-world data. The aim of this research was to evaluate the performance of evolutionary algorithms, and in particular genetic algorithms in large-scale portfolio optimisation. Overall, the results appear to be positive and indicate that genetic algorithms can effectively solve the portfolio optimisation problem in both environments. It has also been observed that genetic algorithms are able to handle larger investment universes better than the MIQP solver.

All three algorithms have been coded in Python 3.9.7. with Gurobi (version 10.0) Python interface. The computational results have been produced by running the algorithms on a Windows computer (AMD Ryzen 7, 1700X 8-Core, 3.4GHz, 48GB RAM) for both data sets.

V.1. Discussion of optimal solution

This section summarises the results obtained from performing the optimisation on the simulated data set. The aim of both algorithms was to optimise the rational investor's objective function comprising of the mean-variance optimisation formula adjusted for the investor's risk-aversion. Before turning to discussing the performance comparison between the two methods, I will review the results of the GA and the MIP solver separately.

V.1.1. Results from the Genetic Algorithm

The main outputs generated by the GA are displayed in Table 3. and Table 4., respectively, to enable a comparison between small and large problem sizes. Apart from the average fitness, the wall-clock time usage during the evaluation was also recorded, as well as the number of generations the GA iterated through. The tables contain the averages taken from the values of the 10 simulations for each problem setting. While Table 3. provides an overview of the results from the smallest investment universe ($N = 500$), Table 4. demonstrates the performance of the GA with the maximum value of the parameter ($N = 5000$).

Closer inspection of the tables shows that the average objective value reached by the GA decreased with both the relaxation of cardinality constraints as well the extension of the time limit. This translates to a better fitness value for both displayed investment universes and can be observed across the other values of N as well. It is important to note that in case of this analysis, a positive objective value denotes a portfolio where the variance exceeds the expected returns, and it is the negative objective value that is desirable by the investor.

Maximum assets (K)	Time limit (T)	Objective value	Wall time* (seconds)	Generations
15	10	0.01744	2.3	2392
	15	0.01658	3.5	3629
	30	0.01735	6.9	7323
	60	0.01666	13.9	14607
100	10	-0.00556	6.2	660
	15	-0.00697	9.3	1069
	30	-0.00881	18.6	2167
	60	-0.00961	37.0	4273
250	10	-0.00788	7.3	148
	15	-0.00847	11.0	271
	30	-0.00921	21.9	640
	60	-0.00983	43.8	1380

Table 3. Average results of the Genetic Algorithm for $N = 500$. A lower objective value translates to a better solution. *Wall time refers to the wall-clock time elapsed during the evaluation of individuals.
The table has been created by the author.

Maximum assets (K)	Time limit (T)	Objective value	Wall time* (seconds)	Generations
15	10	0.02662	1,3	1191
	15	0.02479	1,9	1810
	30	0.02288	3.8	3643
	60	0.02367	7.4	7254
100	10	-0.00349	4,7	470
	15	-0.00608	6,9	746
	30	-0.00921	13.7	1562
	60	-0.01267	27.3	3200
250	10	-0.00655	6.7	109
	15	-0.00736	10.0	209
	30	-0.00970	19.8	507
	60	-0.01255	39.1	1111

Table 4. Average results of the Genetic Algorithm for $N = 5000$. A lower objective value translates to a better solution. *Wall time refers to the wall-clock time elapsed during the evaluation of individuals.
The table has been created by the author.

The tables also reveal that there is a strong negative correlation between the size of the investment universe and the number of generations. In particular, adding more assets to the

investment universe deteriorates the number of generations that can be reached in the same amount of time, with the population losing approximately 6% on average from the generations the algorithm was able to reach for each incremental rise of the problem size. Meanwhile, relaxing the cardinality constraints and allowing more securities to be chosen in the portfolio on average also decrease the number of generations for each time limit.

The third notable observation of the GA results is that increasing the number of assets chosen in the portfolio necessitates a longer evaluation of the fitness of the individuals. While the fitness function only uses a total of around 15-20% of the allowed runtime when $K = 15$, this number increases to 65%-75% when $K = 250$. Therefore, finding the optimal number of assets to be chosen in the portfolio is crucial to improving the efficiency of the genetic algorithm.

Turning now to the performance of the GA from a broader perspective, the results indicate that genetic algorithms can work better in environments where the cardinality constraint is more relaxed. This is demonstrated on Figure 7. across different sizes of the universe of investable securities. The chart on the left-hand side shows an upward trend in average fitness when only 15 assets are chosen in the portfolio. As prior studies have also pointed out, picking only 15 securities might not be sufficient to create a diversified portfolio. This view is also supported by the portfolios created by the GA as none of the average objective function values were in the negative territory for the 15-asset case. In contrast, when the size of the portfolio was fixed at 250, the average fitness was negative, indicating that the expected returns exceeded the variance of the portfolio. Moreover, in this environment the algorithm was able to reach lower fitness for the increasing values of N .

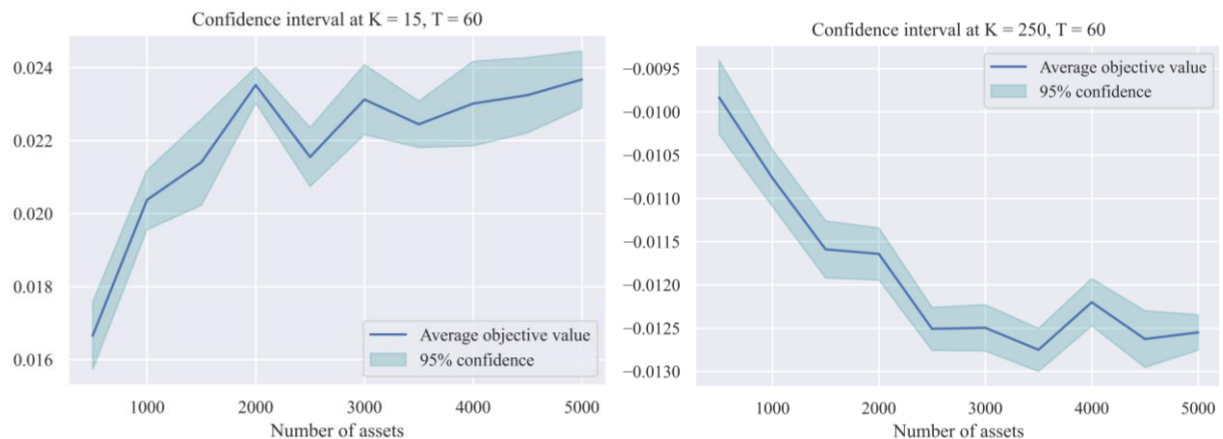


Figure 8. Comparison of average objective value achieved by the GA with 95% confidence interval for different portfolio sizes (K). A lower objective value translates to a better solution. The figure has been created by the author.

V.1.2. Results from the MIQP solver

The next section provides a summary of the results obtained from running the MIQP solver on the same data set that was created for the GA. The average values of the objective function, the wall-clock time and the CPU time are highlighted in Table 5. for $N = 500$, and in Table 6. for $N = 5000$. The most apparent observation from the MIQP optimisation results is the consistency of performance. Given that it is a deterministic optimisation method, allowing additional runtime for the algorithm not result in a noticeable deviation from the previously reached value but rather a higher quality solution. This finding is consistent with that of (Kuendee and Janjarassuk, 2018) who studied MILP and GAs to solve binary problems and concluded that the former provided “*more stable results*”.

Maximum assets (K)	Time limit (T)	Objective value	Wall time* (seconds)	CPU time (seconds)
15	10	0.01331	10.1	55.8
	15	0.01291	15.1	88.6
	30	0.01236	30.1	253.5
	60	0.01217	60.1	660.8
100	10	-0.00992	10.1	63.3
	15	-0.00993	15.1	131.1
	30	-0.00993	30.1	348.5
	60	-0.00993	60.0	793.6
250	10	-0.01006	9.8	33.2
	15	-0.01006	13.0	54.3
	30	-0.01006	19.3	97.1
	60	-0.01006	23.2	124.1

Table 5. Average results of the MIQP algorithm for $N = 500$. A lower objective value translates to a better solution. *The Gurobi solver may exceed the given time limit as the optimisation only stops once the necessary extra computations associated with the termination are finished. The table has been created by the author

While the performance of the MIP solver is similar to the GA in that it finds better solutions with more relaxed cardinality constraint and more time allowance, from the perspective of universe size the performance is quite contrasting. In case of the larger problem sizes such as in Table 6., the MIP algorithm fails to find a profitable solution in most cases. For the investment universe containing 5000 assets, out of the total 120 instances across different time limits and cardinality constraints, 81 returned positive values for the objective. While the value of the

optimal solution also depends on the randomly created returns and the covariance matrix, the ratio of positive objective values for the genetic algorithm was only 33% for $K = 5000$. After carefully reviewing the results, this suggests a significant difference between the objective values at large-scale problems, which is also tested with the hypotheses constructed in IV.4.2.

Maximum assets (K)	Time limit (T)	Objective value	Wall time* (seconds)	CPU time (seconds)
15	10	0.74725	11.7	29.3
	15	0.74725	16.4	43.5
	30	0.74725	30.5	87.0
	60	0.74725	60.5	177.1
100	10	0.74725	11.8	29.9
	15	0.74725	16.4	43.2
	30	0.06484	30.5	86.0
	60	-0.01345	61.6	141.8
250	10	0.74439	13.6	30.0
	15	0.74725	16.4	36.8
	30	-0.01641	35.6	77.2
	60	-0.01725	68.5	119.6

Table 6. Average results of the MIQP algorithm for $N = 5000$. A lower objective value translates to a better solution. *The Gurobi solver may exceed the given time limit as the optimisation only stops once the necessary extra computations associated with the termination are finished. The table has been created by the author.

Another important difference between the GA and MIQP algorithms is their time utilisation for finding the optimal objective. Whereas the time used by the GA is allocated between the different stages of the evolution process and only a fraction is devoted to mathematical evaluation, the MIQP spends the entire time allowance on executing numerical optimisation. Hence, the average of the wall-clock time mostly equals to the time limit and is only smaller when Gurobi finds the optimal solution earlier. On the other hand, the CPU time utilisation of the MIP solver is rather excessive, in several cases it surpasses the wall-clock time by multiple times.

V.2. GA and MIQP comparison

The comparison of the two algorithms and the results of the hypotheses are detailed in this section. The performance of the GA and the MIQP solver is also evaluated against the unconstrained solution, which represents the problem without cardinality constraints.

V.2.1. Discussion of findings

While the results are relatively mixed, they are generally supportive of genetic algorithms and corroborated the assumption that GAs can provide better solutions in large-scaled environments under a given time boundary. Compared to the unconstrained solution, both methods provided worse objective function values as it was expected, however, one unanticipated result was that the unconstrained objective value ultimately increased in the larger investment universes. This finding is likely to be related to the runtime restriction of the unconstrained algorithm as the elapsed wall-clock time only reached 60 seconds when the investment universe contained 4000 or more assets. It can therefore be assumed that the solutions found in the smaller universes represent the global optimum for the unconstrained case.

The results are visualised in Figures 8. and 9. comparing the objective values between the strict and the more relaxed cardinality constraints. While Figure 8. contains the values for $T = 10$, Figure 9. displays the results with the maximum time allowance of 60 seconds. What stands out in almost all charts are the average objective values for the MIQP solver, which far exceed those of the GA. On Figure 8. it can be seen that within the narrow time boundary the GA performed better for the increasing values of N . While the solutions are similar for the smaller investment universes, the difference becomes quite apparent for the larger problem sizes, especially for the tight cardinality constraint of 15 assets. When 250 securities are chosen in the portfolio, the performance of the GA and the MIQP solver are similar and very close to the unconstrained solution under 3000 assets, and the wide gap appears for $N > 3000$. It is worth to mention that the average objective function values for the unconstrained solution are the same across both charts as the problem is solved without any cardinality constraints.

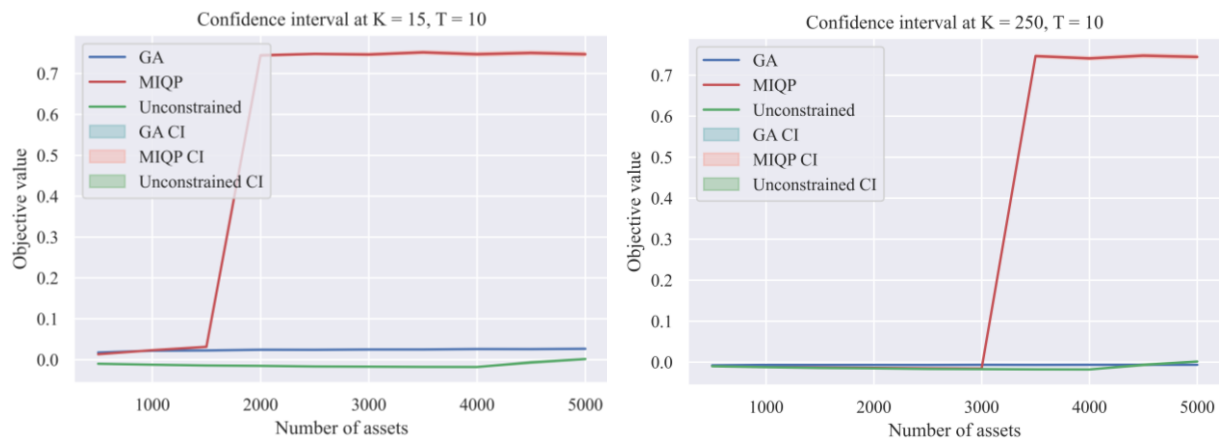


Figure 9. Comparison of average objective values with 90% confidence interval between each algorithm for different portfolio sizes (K) under strict time limit. A lower objective value translates to a better solution. The figure has been created by the author.

The same trend can be observed for the $K, T = \{15, 60\}$ environment on Figure 9, however, the $K, T = \{250, 60\}$ case is significantly different. The results suggest that in a more loosely constrained environment under 60 seconds of time allowance, the MIQP is able to find better solutions than the GA for all problem sizes. However, these results need to be interpreted with caution as the scale of the two charts are different and on the right-hand side the differences between the objective values are much smaller than in any of the previous cases. This is also the reason why, despite the fact that the average objective function values of the unconstrained solution are the same in both figures, their appreciation becomes much more pronounced in the second environment.

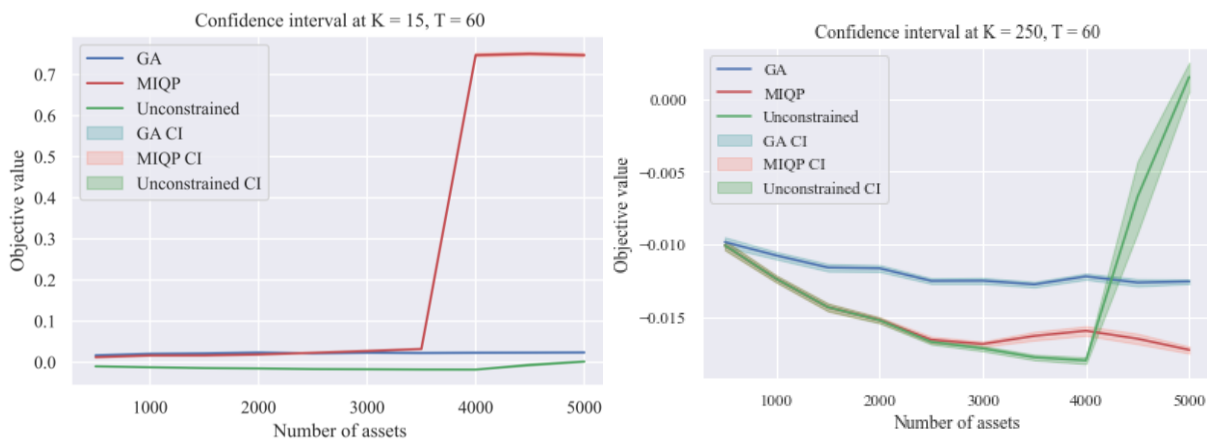


Figure 10. Comparison of average objective values with 90% confidence interval between each algorithm for different portfolio sizes (K) under broader time limit. A lower objective value translates to a better solution. The figure has been created by the author.

V.2.2. Results of the hypothesis testing

This section presents the results of the hypothesis testing, which broadly confirm the findings of the primary analysis. Out of the total 120 different parameter combinations (environment), the hypothesis testing confirmed a significant difference between the average objective function values of the GA and the MIQP approaches in 109 cases for all typical significance levels⁶. In 4 other cases, the difference was only statistically significant on the 5% and 10% significance levels.

Turning to the hypotheses constructed in section IV.4.2., the first statistical analysis was to examine the difference between the average objective values in the smaller asset universe, whereas the second hypothesis was concerned about the larger problem size. The summary of the results are presented in Table 7. and Table 8., whereas the complete list of results is

⁶ The significance level is typically set to 1%, 5% or 10%.

demonstrated in Appendix 2. It is apparent from Table 7. that the GA is not able to reach better objective values than the MIQP solver in a small-sized problem, that is, a universe of 500 assets. All 12 null-hypotheses are rejected in the favour of the MIQP algorithm for all typical significance levels, indicating that the difference between the average objective function values is statistically significant and that the deterministic approach can reach a lower objective value than the GA. The implication of this finding is that for a smaller universe of assets it is preferable to optimise the portfolio using mixed integer quadratic programming.

Maximum assets (K)	Time limit (T)	Z-value	P-value	Testing result
15	10	55	0.00098	H_0 rejected
	15	55	0.00098	H_0 rejected
	30	55	0.00098	H_0 rejected
	60	55	0.00098	H_0 rejected
100	10	55	0.00098	H_0 rejected
	15	55	0.00098	H_0 rejected
	30	55	0.00098	H_0 rejected
	60	55	0.00098	H_0 rejected
250	10	55	0.00098	H_0 rejected
	15	55	0.00098	H_0 rejected
	30	55	0.00098	H_0 rejected
	60	55	0.00098	H_0 rejected

Table 7. Results of the right-tailed Wilcoxon signed-rank test for $N = 500$. $H_0: U(GA) < U(MIQP)$.
The table has been created by the author.

As opposed to the small-asset case, the results from the large-scale optimisation problem in Table 8. are more in favour of the GA algorithm. The nullhypothesis was rejected in 8 environments for all typical significance levels, implying that there is a high possibility that the average objective function values of the GA are in fact lower than the averages of the MIQP solver. However, the GA was not successful in all environments, in particular the null-hypotheses were accepted in those cases when the cardinality constraint was more relaxed and the time limit was allowing a longer runtime.

Maximum assets (K)	Time limit (T)	Z-value	P-value	Testing result
15	10	0	0.00098	H_0 rejected
	15	0	0.00098	H_0 rejected
	30	0	0.00098	H_0 rejected
	60	0	0.00098	H_0 rejected
100	10	0	0.00098	H_0 rejected
	15	0	0.00098	H_0 rejected
	30	45	0.96777	H_0 accepted*
	60	51	0.99512	H_0 accepted
250	10	0	0.00098	H_0 rejected
	15	0	0.00098	H_0 rejected
	30	55	1.00000	H_0 accepted
	60	55	1.00000	H_0 accepted

Table 8. Results of the left-tailed Wilcoxon signed-rank test for $N = 5000$. $H_0: U(GA) > U(MIQP)$

*When tested with two-tailed Wilcoxon signed-rank test, the null hypothesis was accepted on the 10% significance level, indicating no statistically significant difference between the mean objective values.

The table has been created by the author.

The results are similar for other levels of N with an overall upward trend in the performance of the GA as the universe of investable assets increased. While for the first two sizes no GA solutions were able to compete with the MIQP solver, the number of statistically significant differences in favour of the GA gradually increased in line with the growth of the universe. As demonstrated in Figure 10., the proportion of GA runs outperforming the MIQP solutions reaches approximately 67% for the 5000-asset universe.

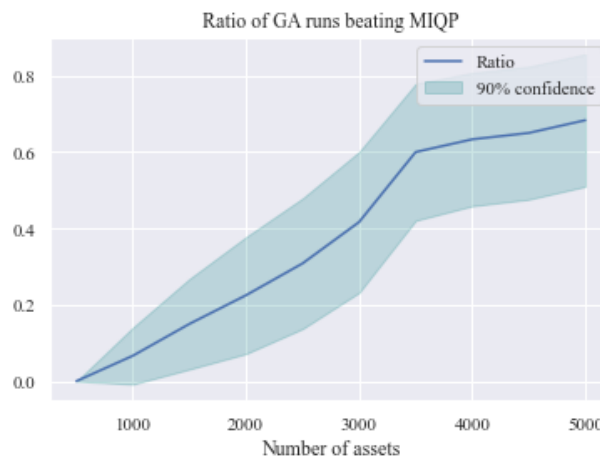


Figure 11. Performance improvement of the GA for increasing problem size with 90% confidence interval for aggregated results. The figure has been created by the author.

V.3. Results of testing with market data

In the final part of the analysis, I examine the results from running the algorithms on the data set exported from Yahoo Finance. While the overall results are similar to those of the simulation, there are a number of interesting aspects observed in the real-data environment. In general, it can be concluded that the algorithms behave in the same manner, and their performance has also been broadly consistent with the simulated outcome.

The most interesting results of running the algorithms in the real-world environment is that while the GA shows a relatively robust performance, the solutions delivered by the MIQP have deteriorated considerably. From the data in Figure 12., it is apparent that in the real-data setting, Gurobi is much less able to keep up with the performance of the GA or the unconstrained solution when it is subject to a time boundary. Within a time allowance of 60 seconds, the MIQP solver can only approximate the optimal solution effectively for universes under 3000 assets, for larger problem sizes the 60 seconds are not enough for Gurobi to arrive near the optimal solution. When given a time boundary of 10 seconds, the MIQP algorithm can only deliver good solutions until 1500 assets in the universe. The same results can be observed for different relaxations of the cardinality constraint as well, however, the differences are much less noticeable between different values of K . These findings suggest that despite the covariance was corrected through the shrinkage method, the MIQP solver performs less effectively in the real data environment than it does in the simulation, especially when given a larger problem size.

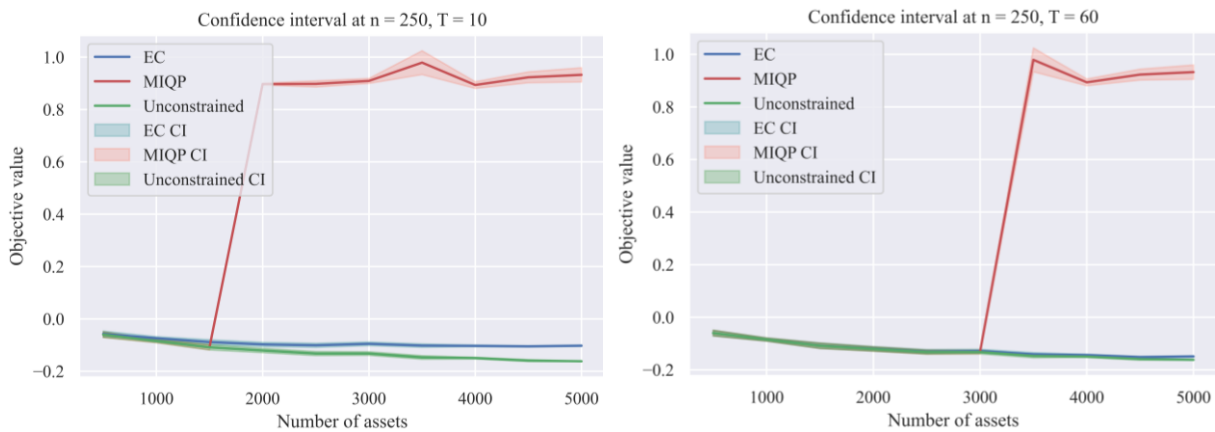


Figure 12. Comparison of average objective values for real data set with 90% confidence for different time limits (T). A lower objective value translates to a better solution. The figure has been created by the author.

The second notable aspect of the market data experiment is the performance of the GA in relation to the MIQP algorithm. Figure 13. compares the results obtained from running the

algorithms in the simulated and the real-data environment and in particular highlights the percentage of GA runs that beat the MIQP solution throughout the different problem sizes. While the simulated environment results in a gradual increase in the GA's success ratio, the results generated with the market data are rather skewed towards the extreme values. For the first three sizes, no average GA objective values are as good as the averages of the MIQP. However, after the 3000-asset size the ratio jumps to 100%, meaning that all average GA objective values were lower than the values created by Gurobi.

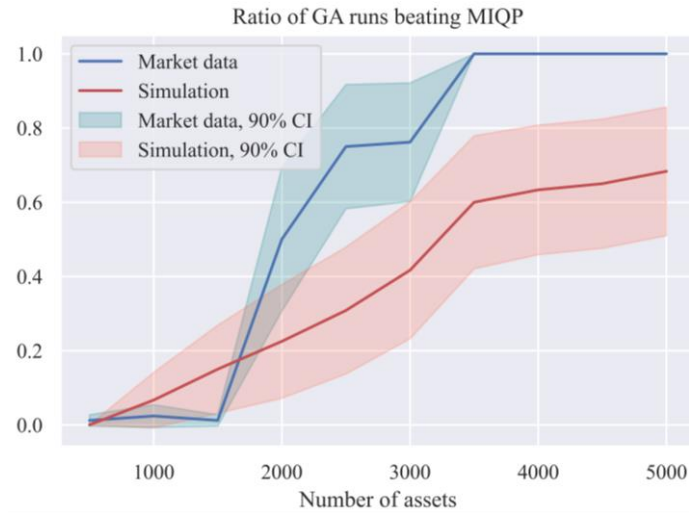


Figure 13. Comparison of performance improvement for aggregated results of the GA in the simulated and real environment for increasing problem size with 90% confidence interval. The figure has been created by the author.

In order to test the significance of the difference between the average objective function values of the two methods, the same hypothesis testing was conducted for the market data results using the Wilcoxon signed-rank test. It is worth to mention that the use of a nonparametric test was much more reasonable than in the simulated environment as a higher proportion of test results did not follow a normal distribution. Table 9. and Table 10. demonstrate the results of the hypotheses for the same problem settings as those in Figure 12, that is, a comparison between $K, T = \{250, 10\}$ and $K, T = \{250, 60\}$ is enabled by the data.

The results of the hypothesis testing are consistent with the data visualised in Figure 13 and are also in line with the findings of the simulated environment. The null-hypotheses can be rejected in the large-scale cases for both variable combinations, suggesting that the difference between the average objective function values of the GA and the MIQP is statistically significant, and more specifically, the values of the GA are lower. The tables also reveal that in the narrower timeframe the solutions of the GA reach the level of significant difference much faster, whereas

in case of the 60-second runtime, only the average objective function values above 3000 assets are meaningfully lower.

Problem size (N)	GA average	MIQP average	P-value	Testing result
500	-0.05768	-0.06023	1.00000	H_0 accepted
1000	-0.07398	-0.08487	1.00000	H_0 accepted
1500	-0.08863	-0.10854	1.00000	H_0 accepted
2000	-0.09761	0.89674	0.00781	H_0 rejected
2500	-0.10099	0.89756	0.00781	H_0 rejected
3000	-0.09532	0.90897	0.00781	H_0 rejected
3500	-0.10208	0.97871	0.00781	H_0 rejected
4000	-0.10329	0.89330	0.00781	H_0 rejected
4500	-0.10506	0.92274	0.00781	H_0 rejected
5000	-0.10262	0.93215	0.00781	H_0 rejected

Table 9. Results of the left-tailed Wilcoxon signed-rank test with the maximum assets (K) to be chosen is 250 and the time limit (T) is 10. $H_0: U(\text{GA}) > U(\text{MIQP})$. A lower objective value translates to a better solution. The table has been created by the author.

Problem size (N)	U(GA) average	U(MIQP) average	P-value	Testing result
500	-0.06023	-0.06023	1.00000	H_0 accepted
1000	-0.08466	-0.08487	1.00000	H_0 accepted
1500	-0.10791	-0.10854	1.00000	H_0 accepted
2000	-0.11975	-0.12084	1.00000	H_0 accepted
2500	-0.13094	-0.13254	1.00000	H_0 accepted
3000	-0.12853	-0.13253	1.00000	H_0 accepted
3500	-0.14114	0.97872	0.00781	H_0 rejected
4000	-0.14409	0.89330	0.00781	H_0 rejected
4500	-0.15181	0.92274	0.00781	H_0 rejected
5000	-0.14925	0.93215	0.00781	H_0 rejected

Table 10. Results of the left-tailed Wilcoxon signed-rank test with the maximum assets (K) to be chosen is 250 and the time limit (T) is 60. $H_0: U(\text{GA}) > U(\text{MIQP})$. A lower objective value translates to a better solution. The table has been created by the author.

VI. Conclusion and future work

This study was designed to evaluate the performance of genetic algorithms in a cardinality and time constrained portfolio optimisation problem. Cardinality constraints induce a more difficult formulation of the optimisation problem as they restrict the number of assets that are allowed in the portfolio. As the search of literature revealed few studies concerned with this subject, part of the research was to review GAs in comparison to the MIQP solver Gurobi, which represents the deterministic optimisation method to solve the given portfolio optimisation problem. Both algorithms were evaluated in relation to the unconstrained solution as well, which did not have a limit on the number of assets that can be chosen in the portfolio from the total asset universe. The algorithms were tested in both a simulated environment with randomly generated expected returns and covariance matrix, and a real-world environment that contained market data from 5200 stocks traded on the New York Stock Exchange.

The results of this investigation show that GAs are generally effective in solving the cardinality constrained portfolio optimisation problem, and they are able to outperform the MIQP solver in large investment universes. While the algorithm did not always deliver the best solutions across different environments, its performance is consistent, and it is able to compete with the MIQP. The testing in the simulated environment strengthened the general view of the robustness of the algorithm, however, the findings are limited in both the problem definition and the parameters of the algorithms. A greater focus on practical constraints such as transaction costs could greatly improve the utilisation of this work in real-life problems.

The comparison between the two methods revealed a superiority of GAs in most large-scale environments, particularly in the presence of tighter cardinality constraints and time boundary. For small asset cases, the deterministic approach remains better in finding solutions. However, it is also worth to mention that Gurobi and similar mixed integer solvers require a licence and are generally not free to use, whereas genetic algorithms can be constructed freely and therefore provide a good alternative to commercialised software.

When testing in a real-world environment, it can be concluded that GAs are able to create diversified portfolios in real-life situations. Moreover, their performance is considerably better than that of the MIQP, which showed a less satisfactory performance for increasing problem sizes. These findings point to several practical implications of GAs to create various investment strategies. Enhanced versions of this GA can potentially be used for index tracking or portfolio optimisation and rebalancing across several time periods.

While the overall results are supportive of the performance of GAs, their practical use remains limited as they are computationally expensive through their extensive data usage and significant runtime. Possible improvements of the algorithm include higher selection pressure and the mutation of all offspring, which could eventually increase convergence towards the optimal solution and decrease the runtime requirement of the algorithm.

The findings of this research provide a deeper insight into the performance of GAs for cardinality constrained portfolio optimisation problems. In spite of their limitations, GAs are quite effective at solving these problems and generally provide good solutions in the presence of time constraints as well. Their easy implementation makes them a beneficial method for portfolio optimisation and through continued efforts they may also be applied to real-life situations.

VII. Bibliography

- Back, T., Hammel, U. and Schwefel, H.-P. (1997) 'Evolutionary computation: comments on the history and current state', *IEEE Transactions on Evolutionary Computation*, 1(1), pp. 3–17. Available at: <https://doi.org/10.1109/4235.585888>.
- Baker, J.E. (1987) 'Reducing bias and inefficiency in the selection algorithm', *Genetic algorithms and their applications : proceedings of the second International Conference on Genetic Algorithms : July 28-31, 1987 at the Massachusetts Institute of Technology, Cambridge, MA* [Preprint]. Available at: https://scholar.google.com/scholar_lookup?title=Reducing+bias+and+inefficiency+in+the+selection+algorithm&author=Baker%2C+J.E.&publication_year=1987 (Accessed: 4 April 2023).
- Bertsimas, D., Darnell, C. and Soucy, R. (1999) 'Portfolio Construction Through Mixed-Integer Programming at Grantham, Mayo, Van Otterloo and Company', *Interfaces*, 29(1), pp. 49–66. Available at: <https://doi.org/10.1287/inte.29.1.49>.
- Bodie, Z., Kane, A. and Marcus, A.J. (2014) *Investments*. 10. ed. New York, NY: Mc-Graw-Hill Education (The McGraw-Hill/Irwin series in finance, insurance and real estate).
- Canakgoz, N.A. and Beasley, J.E. (2009) 'Mixed-integer programming approaches for index tracking and enhanced indexation', *European Journal of Operational Research*, 196(1), pp. 384–399. Available at: <https://doi.org/10.1016/j.ejor.2008.03.015>.
- Cavazzuti, M. (2013) *Optimization Methods: From Theory to Design Scientific and Technological Aspects in Mechanics*. Berlin, Heidelberg: Springer. Available at: <https://doi.org/10.1007/978-3-642-31187-1>.
- Chang, T.-J. *et al.* (2000) 'Heuristics for cardinality constrained portfolio optimisation', *Computers & Operations Research*, 27(13), pp. 1271–1302. Available at: [https://doi.org/10.1016/S0305-0548\(99\)00074-X](https://doi.org/10.1016/S0305-0548(99)00074-X).
- Costa, L. and Oliveira, P. (2001) 'Evolutionary algorithms approach to the solution of mixed integer non-linear programming problems', *Computers & Chemical Engineering*, 25(2), pp. 257–266. Available at: [https://doi.org/10.1016/S0098-1354\(00\)00653-0](https://doi.org/10.1016/S0098-1354(00)00653-0).
- Dreżewski, R., Dziuban, G. and Pająk, K. (2018) 'The Bio-Inspired Optimization of Trading Strategies and Its Impact on the Efficient Market Hypothesis and Sustainable Development Strategies', *Sustainability*, 10(5), p. 1460. Available at: <https://doi.org/10.3390/su10051460>.
- Eiben, A.E. and Smit, S.K. (2012) 'Evolutionary Algorithm Parameters and Methods to Tune Them', in Y. Hamadi, E. Monfroy, and F. Saubion (eds) *Autonomous Search*. Berlin, Heidelberg: Springer, pp. 15–36. Available at: https://doi.org/10.1007/978-3-642-21434-9_2.
- Eiben, A.E. and Smith, J.E. (2015) *Introduction to Evolutionary Computing*. 2nd edn. Springer Publishing Company, Incorporated.

- Elton, E.J. and Gruber, M.J. (1997) ‘Modern portfolio theory, 1950 to date’, *Journal of Banking & Finance*, 21(11), pp. 1743–1759. Available at: [https://doi.org/10.1016/S0378-4266\(97\)00048-4](https://doi.org/10.1016/S0378-4266(97)00048-4).
- Fabozzi, F.J. *et al.* (2007) *Robust Portfolio Optimization and Management*. 1st edition. Hoboken, New Jersey: Wiley.
- Fama, E.F. (1970) ‘Multiperiod Consumption-Investment Decisions’, *The American Economic Review*, 60(1), pp. 163–174.
- Fama, E.F. and French, K.R. (1992) ‘The Cross-Section of Expected Stock Returns’, *The Journal of Finance*, 47(2), pp. 427–465. Available at: <https://doi.org/10.2307/2329112>.
- Fieldsend, J.E., Matatko, J. and Peng, M. (2004) ‘Cardinality Constrained Portfolio Optimisation’, in Z.R. Yang, H. Yin, and R.M. Everson (eds) *Intelligent Data Engineering and Automated Learning – IDEAL 2004*. Berlin, Heidelberg: Springer (Lecture Notes in Computer Science), pp. 788–793. Available at: https://doi.org/10.1007/978-3-540-28651-6_117.
- Goldberg, D.E. (1989) *Genetic Algorithms in Search, Optimization and Machine Learning*. 13th ed. edition. Reading, Mass: Addison-Wesley Professional.
- Guo, Q. (2022) ‘Review of Research on Markowitz Model in Portfolios’, in. *2022 7th International Conference on Social Sciences and Economic Development (ICSSED 2022)*, Atlantis Press, pp. 786–790. Available at: <https://doi.org/10.2991/aebmr.k.220405.131>.
- Hochba, D.S. (1997) ‘Approximation Algorithms for NP-Hard Problems’, *ACM SIGACT News*, 28(2), pp. 40–52. Available at: <https://doi.org/10.1145/261342.571216>.
- Holland, J.H. (1975) *Adaptation in natural and artificial systems: An introductory analysis with applications to biology, control, and artificial intelligence*. Oxford, England: U Michigan Press (Adaptation in natural and artificial systems: An introductory analysis with applications to biology, control, and artificial intelligence), pp. viii, 183.
- Hull, J.C. (2018) *Risk Management and Financial Institutions*. 5th edition. Hoboken, New Jersey: Wiley.
- Jobson, J.D. and Korkie, B. (1980) ‘Estimation for Markowitz Efficient Portfolios’, *Journal of the American Statistical Association*, 75(371), pp. 544–554. Available at: <https://doi.org/10.2307/2287643>.
- Katoch, S., Chauhan, S.S. and Kumar, V. (2021) ‘A review on genetic algorithm: past, present, and future’, *Multimedia Tools and Applications*, 80(5), pp. 8091–8126. Available at: <https://doi.org/10.1007/s11042-020-10139-6>.
- Kolm, P.N., Tütüncü, R. and Fabozzi, F.J. (2014) ‘60 Years of portfolio optimization: Practical challenges and current trends’, *European Journal of Operational Research*, 234(2), pp. 356–371. Available at: <https://doi.org/10.1016/j.ejor.2013.10.060>.
- Kuendee, P. and Janjarassuk, U. (2018) ‘A comparative study of mixed-integer linear programming and genetic algorithms for solving binary problems’, in *2018 5th International Conference on Industrial Engineering and Applications (ICIEA)*. 2018 5th

- International Conference on Industrial Engineering and Applications (ICIEA)*, pp. 284–288. Available at: <https://doi.org/10.1109/IEA.2018.8387111>.
- Lazimy, R. (1982) ‘Mixed-integer quadratic programming’, *Mathematical Programming*, 22(1), pp. 332–349. Available at: <https://doi.org/10.1007/BF01581047>.
- Ledoit, O. and Wolf, M. (2004) ‘Honey, I Shrunk the Sample Covariance Matrix’, *The Journal of Portfolio Management*, 30(4), pp. 110–119. Available at: <https://doi.org/10.3905/jpm.2004.110>.
- Mansini, R., Ogryczak, W. and Speranza, M.G. (2015) *Linear and Mixed Integer Programming for Portfolio Optimization*, *Linear and Mixed Integer Programming for Portfolio Optimization*, p. 119. Available at: <https://doi.org/10.1007/978-3-319-18482-1>.
- Markowitz, H. (1952) ‘Portfolio Selection’, *The Journal of Finance*, 7(1), pp. 77–91. Available at: <https://doi.org/10.2307/2975974>.
- Markowitz, H.M. (1959) *Portfolio Selection: Efficient Diversification of Investments*. Yale University Press. Available at: <https://www.jstor.org/stable/j.ctt1bh4c8h> (Accessed: 25 April 2022).
- Merton, R.C. (1969) ‘Lifetime Portfolio Selection under Uncertainty: The Continuous-Time Case’, *The Review of Economics and Statistics*, 51(3), pp. 247–257. Available at: <https://doi.org/10.2307/1926560>.
- Merton, R.C. (1972) ‘An Analytic Derivation of the Efficient Portfolio Frontier’, *The Journal of Financial and Quantitative Analysis*, 7(4), p. 1851. Available at: <https://doi.org/10.2307/2329621>.
- MOSEK ApS (2022) ‘MOSEK Modeling Cookbook’.
- Nemati, M., Braun, M. and Tenbohlen, S. (2018) ‘Optimization of unit commitment and economic dispatch in microgrids based on genetic algorithm and mixed integer linear programming’, *Applied Energy*, 210, pp. 944–963. Available at: <https://doi.org/10.1016/j.apenergy.2017.07.007>.
- Nocedal, J. and Wright, S.J. (2006) *Numerical optimization*. Second edition. New York, NY: Springer (Springer series in operation research and financial engineering).
- Ouaïl, F. and Chergui, M. (2018) ‘A branch-and-cut technique to solve multiobjective integer quadratic programming problems’, *Annals of Operations Research*, 267. Available at: <https://doi.org/10.1007/s10479-017-2698-6>.
- Peters, J. *et al.* (2019) ‘Mixed Integer Programming versus Evolutionary Computation for Optimizing a Hard Real-World Staff Assignment Problem’, *Proceedings of the International Conference on Automated Planning and Scheduling*, 29, pp. 541–554. Available at: <https://doi.org/10.1609/icaps.v29i1.3521>.
- Pia, A.D., Dey, S.S. and Molinaro, M. (2017) ‘Mixed-integer quadratic programming is in NP’, *Mathematical Programming*, 162(1), pp. 225–240. Available at: <https://doi.org/10.1007/s10107-016-1036-0>.

- Pogue, G.A. (1970) 'An Extension of the Markowitz Portfolio Selection Model to Include Variable Transactions' Costs, Short Sales, Leverage Policies and Taxes', *The Journal of Finance*, 25(5), pp. 1005–1027. Available at: <https://doi.org/10.1111/j.1540-6261.1970.tb00865.x>.
- Prigent, J.-L. (2008) *Portfolio Optimization and Performance Analysis*. New York: Chapman and Hall/CRC. Available at: <https://doi.org/10.1201/9781420010930>.
- Roll, R. and Ross, S.A. (1980) 'An Empirical Investigation of the Arbitrage Pricing Theory', *The Journal of Finance*, 35(5), pp. 1073–1103. Available at: <https://doi.org/10.2307/2327087>.
- Rothberg, E. (2007) 'An Evolutionary Algorithm for Polishing Mixed Integer Programming Solutions', *INFORMS Journal on Computing*, 19(4), pp. 534–541. Available at: <https://doi.org/10.1287/ijoc.1060.0189>.
- Roussel, O. (2011) 'Controlling a Solver Execution with the runsolver Tool: System description', *Journal on Satisfiability, Boolean Modeling and Computation*, 7(4), pp. 139–144. Available at: <https://doi.org/10.3233/SAT190083>.
- Roy, A.D. (1952) 'Safety First and the Holding of Assets', *Econometrica*, 20(3), pp. 431–449. Available at: <https://doi.org/10.2307/1907413>.
- Schaffer, J. (1985) 'Multiple Objective Optimization with Vector Evaluated Genetic Algorithms.', in: *Proceedings of the First Int. Conference on Genetic Algorithms*, Ed. G.J.E Grefenstette, J.J. Lawrence Erlbaum, pp. 93–100.
- Schwefel, H.-P. (2000) 'Advantages (and Disadvantages) of Evolutionary Computation Over Other Approaches', *Evol Comput*, 1.
- Scott, E.O. and De Jong, K.A. (2015) 'Understanding Simple Asynchronous Evolutionary Algorithms', in: *Proceedings of the 2015 ACM Conference on Foundations of Genetic Algorithms XIII. FOGA '15: Foundations of Genetic Algorithms XIII*, Aberystwyth United Kingdom: ACM, pp. 85–98. Available at: <https://doi.org/10.1145/2725494.2725509>.
- Sharpe, W.F. (1963) 'A Simplified Model for Portfolio Analysis', *Management Science*, 9(2), pp. 277–293.
- Student (1908) 'The Probable Error of a Mean', *Biometrika*, 6(1), pp. 1–25. Available at: <https://doi.org/10.2307/2331554>.
- Wilcoxon, F. (1945) 'Individual Comparisons by Ranking Methods', *Biometrics Bulletin*, 1(6), pp. 80–83. Available at: <https://doi.org/10.2307/3001968>.
- Wolsey, L.A. (2008) 'Mixed Integer Programming', in: *Wiley Encyclopedia of Computer Science and Engineering*. John Wiley & Sons, Ltd, pp. 1–10. Available at: <https://doi.org/10.1002/9780470050118.ecse244>.
- Yoshimoto, A. (1996) 'The Mean-Variance Approach to Portfolio Optimization Subject to Transaction Costs', *日本オペレーションズ・リサーチ学会論文誌*, 39(1), pp. 99–117. Available at: <https://doi.org/10.15807/jorsj.39.99>.

- Zaimovic, A., Omanovic, A. and Arnaut-Berilo, A. (2021) ‘How Many Stocks Are Sufficient for Equity Portfolio Diversification? A Review of the Literature’, *Journal of Risk and Financial Management*, 14(11), p. 551. Available at: <https://doi.org/10.3390/jrfm14110551>.
- Zhang, Y., Li, X. and Guo, S. (2018) ‘Portfolio selection problems with Markowitz’s mean–variance framework: a review of literature’, *Fuzzy Optimization and Decision Making*, 17(2), pp. 125–158. Available at: <https://doi.org/10.1007/s10700-017-9266-z>.

Appendix 1.

Appendix 1.1 – Random correlation matrix generation

```
# correlation matrix for input data
def generate_corr_matrix(d):
    for x in range(10):
        # Initialization
        beta = 1.5 + (d - 2.0) / 2.0
        u = np.random.beta(beta, beta)
        r12 = 2.0 * u - 1.0
        r = np.matrix([[1, r12],
                        [r12, 1]])

        # Onion method
        for n in range(2, d):
            beta = beta - 0.5
            y = np.random.beta(n / 2.0, beta)
            u_vec = np.random.normal(size = (n, 1))
            u_vec = u_vec / np.linalg.norm(u_vec, ord = 2)
            w_vec = np.sqrt(y) * u_vec
            A_mat = np.linalg.cholesky(r)
            z = A_mat @ w_vec
            r = np.block([[r, z], [z.T, np.ones(1)]]])

        if min(np.linalg.eigvals(r)) >= 0:
            return r

    return None
```

Appendix 1.2 – Define data generation function

```
# creating input for the algorithm
def get_data(n_assets: int):
    # seed = 100
    # random mean returns generated for each asset, mu=0,
    sigma=0.04
    returns = np.random.normal(0, 0.04, n_assets)
    # randomly created correlation matrix
    correl = generate_corr_matrix(n_assets)

    return returns, correl
```

Appendix 1.3 – Define unconstrained optimisation function

```
#optimisation for weights, without dummy variable
def optimise(n_variables, Lambda, returns, correl):
    model = gp.Model()
    model.Params.LogToConsole = 0
    model.Params.NumericFocus = 1
    model.Params.Presolve = 0
    model.Params.TimeLimit = 60

    w = model.addMVar(n_variables)
    model.update()

    portfoliorisk = w @ correl @ w
    portfolioreturn = returns @ w
    model.setObjective(Lambda * portfoliorisk - (1 - Lambda) *
portfolioreturn, gp.GRB.MINIMIZE)
    model.addConstr(w.sum() == 1)
    model.update()

    start_wall, start_cpu = time.time(), time.process_time()
    model.optimize()
    end_wall, end_cpu = time.time(), time.process_time()

    wall_time_in_gurobi = end_wall - start_wall
    cpu_time_in_gurobi = end_cpu - start_cpu

    optimalobjective = model.objval
    optimalw = w.X

    return optimalobjective, optimalw, wall_time_in_gurobi,
cpu_time_in_gurobi
```

Appendix 1.4 – Define MIQP optimisation function

```
#optimisation for weights, number of chosen assets is
controlled
def optimise_mip_timelimit(n_assets, n_chosen, Lambda, mu,
cov, max_time_sec):
    #setup steps
    z = np.array([True,False])
    M = 1000.0

    #model setup
    model = gp.Model()

    # Parameters
    model.Params.LogToConsole = 0
    model.Params.NumericFocus = 1
    model.Params.TimeLimit = max_time_sec

    w = model.addMVar(len(mu))
    model.update()
    portfoliorisk = w @ cov @ w
    portfolioreturn = mu @ w
    model.setObjective(Lambda * portfoliorisk - (1 - Lambda) *
portfolioreturn, gp.GRB.MINIMIZE)
    model.addConstr(w.sum() == 1)

    z = pd.Series(model.addVars(len(mu), vtype =
gp.GRB.BINARY))
    model.update()

    for i in range(len(mu)):
        model.addConstr(w[i] <= M * z[i])

    model.addConstr(z.sum() == n_chosen)
    model.update()

    start_wall, start_cpu = time.time(), time.process_time()
    model.optimize()
    end_wall, end_cpu = time.time(), time.process_time()

    wall_time_in_gurobi = end_wall - start_wall
    cpu_time_in_gurobi = end_cpu - start_cpu

    optimal_weights = w.X
    optimal_objective = model.objval

    return optimal_objective, optimal_weights,
wall_time_in_gurobi, cpu_time_in_gurobi
```

Appendix 1.5 – Define class that stores results

```
class Solution:
    def __init__(self, n_assets: int):
        self.genes = np.zeros(n_assets)
        self.w = np.zeros(n_assets)
        self.score = -10000.0
        self.wall_time = 0.0
        self.cpu_time = 0.0

    def init_random(self, n_chosen: int):
        self.genes[: n_chosen] = 1

        np.random.shuffle(self.genes)

    def evaluate_EC(self, n_chosen, Lambda, returns, risk):
        selection_return = returns[(np.nonzero(self.genes))]
        selection_risk = risk[self.genes == 1, :][:,
self.genes == 1]
        fitness, optimal_w, wall_time_in_gurobi,
cpu_time_in_gurobi = optimise(n_chosen, Lambda,
selection_return, selection_risk)
        self.score = fitness
        self.w = optimal_w
        self.wall_time = wall_time_in_gurobi
        self.cpu_time = cpu_time_in_gurobi
```

Appendix 1.6 – Define function to select parents and generate offspring

```
def generate_offspring(population, n_assets, n_chosen,
pop_size):
    pool1 = random.sample(range(0, pop_size), 2)
    pool2 = random.sample(range(0, pop_size), 2)

    parent1 = population[pool1[np.argmin(np.array([i.score for
i in population])[pool1])]]
    parent2 = population[pool2[np.argmin(np.array([i.score for
i in population])[pool2])]]

    offspring = []
    randomvar = np.random.uniform(0, 1, n_assets)
    for i in range(n_assets):
        if randomvar[i] >= 0.5:
            offspring.append(parent1.genes[i])
        elif randomvar[i] < 0.5:
            offspring.append(parent2.genes[i])
    offspring = np.array(offspring)

    if sum(offspring) > n_chosen:
        mutate = int(sum(offspring) - n_chosen)
        ones = np.array(np.where(offspring == 1)[0])
        np.random.shuffle(ones)
        for j in range(mutate):
            offspring[ones[j]] = 0
    elif (sum(offspring) < n_chosen):
        mutate = int(n_chosen - sum(offspring))
        zeros = np.array(np.where(offspring == 0)[0])
        np.random.shuffle(zeros)
        for j in range(mutate):
            offspring[zeros[j]] = 1

    return offspring
```

Appendix 1.7 – Define survivor selection

```
#selecting genes for next generation
def next_gen(returns, correl, population, offspring_genes,
n_assets, n_chosen, Lambda):
    offspring = Solution(n_assets)
    offspring.genes = offspring_genes
    offspring.evaluate_EC(n_chosen, Lambda, returns, correl)

    weakest = np.argmax(np.array([i.score for i in
population]))
    population[weakest] = offspring

    return population, offspring.wall_time, offspring.cpu_time
```

Appendix 1.8 – Define genetic algorithm

```
def genetic_algorithm_timelimit(returns, correl, n_assets,
n_chosen, pop_size: int = 100, max_time_sec: float=10.0,
Lambda: float = 0.75):
    #setup
    start_time = time.time()
    generation_counter = 0

    avgscore = []
    total_wall_time = 0
    total_cpu_time = 0

    population = np.empty((0, n_assets))
    for i in range(pop_size):
        temp = Solution(n_assets)
        temp.init_random(n_chosen)
        temp.evaluate_EC(n_chosen, Lambda, returns, correl)
        total_wall_time = total_wall_time + temp.wall_time
        total_cpu_time = total_cpu_time + temp.cpu_time
        population = np.append(population, temp)

    while time.time() - start_time < max_time_sec:
        #parent selection (tournament size k=2, crossover rate
p=0.5 and mutation)
        offspring_genes = generate_offspring(population,
n_assets, n_chosen, pop_size)

        #creating next generation by replacing weakest
chromosome with offspring
        population, wall_time_in_gurobi, cpu_time_in_gurobi =
next_gen(returns, correl, population, offspring_genes,
n_assets, n_chosen, Lambda)
        total_wall_time = total_wall_time +
wall_time_in_gurobi
        total_cpu_time = total_cpu_time + cpu_time_in_gurobi

        #saving best score of current generation
        avgscore.append(population[np.argmin(np.array([i.score
for i in population]))].score)

        generation_counter += 1

    #finding optimal solution from last generation
    optimal_solution = population[np.argmin(np.array([i.score
for i in population]))]

    return optimal_solution, avgscore, total_wall_time,
total_cpu_time, generation_counter
```

Appendix 1.9 – Code to run simulation

```
results_mip = []
results_ec = []
results_unconstrained = []

N_pop = 100 # the size of the population
Lambda = 0.75 # parameter for gurobi optimisation
N_assets_list = [500, 1000, 1500, 2000, 2500, 3000, 3500,
4000, 4500, 5000]

for run in range(10):
    for N_assets in N_assets_list:
        returns, correl = get_data(N_assets)

        print(f'N_assets = {N_assets}')

        # --- UNCONSTRAINED ---
        unconstrained_sol = Solution(N_assets)
        unconstrained_sol.score, unconstrained_sol.w,
unconstrained_sol.wall_time, unconstrained_sol.cpu_time =
optimise(
            N_assets, Lambda, returns, correl)

        results_unconstrained.append({
            'Number of assets': N_assets,
            'Score': unconstrained_sol.score,
            'Wall clock time': unconstrained_sol.wall_time,
            'CPU time': unconstrained_sol.cpu_time
        })

        print('\tUnconstrained ok')

        for max_included_assets in [15, 100, 250]:
            print(f'\tmax_included_assets =
{max_included_assets}')

            for max_time_sec in [10, 15, 30, 60]:
                print(f'\t\tmax_included_assets =
{max_included_assets}')

                # --- GUROBI MIP ---
                mip_sol = Solution(N_assets)
                mip_sol.score, mip_sol.w, mip_sol.wall_time,
mip_sol.cpu_time = optimise_mip_timelimit(
                    N_assets, max_included_assets, Lambda,
                    returns, correl, max_time_sec)

                results_mip.append({
                    'Number of assets': N_assets,
                    'Score': mip_sol.score,
```

```

        'Wall clock time': mip_sol.wall_time,
        'CPU time': mip_sol.cpu_time,
        'max_included_assets':
max_included_assets,
        'max_time_sec': max_time_sec,
    })

    print('\t\tMIP ok.')

    # Solve with EC
    start_wall, start_cpu = time.time(),
time.process_time()
    ec_sol, average_score, ec_wall_time_in_gurobi,
ec_cpu_time_in_gurobi, N_generations =
genetic_algorithm_timelimit(
        returns, correl, N_assets,
max_included_assets, N_pop, max_time_sec, Lambda)
    end_wall, end_cpu = time.time(),
time.process_time()

    ec_wall_time = end_wall - start_wall
    ec_cpu_time = end_cpu - start_cpu
    ec_gurobi_only_wall = ec_wall_time_in_gurobi
    ec_gurobi_only_cpu = ec_cpu_time_in_gurobi

    # Solve unconstrained

    results_ec.append({
        'Number of assets': N_assets,
        'Score': ec_sol.score,
        'Wall clock time': ec_gurobi_only_wall,
        'CPU time': ec_gurobi_only_cpu,
        'max_included_assets':
max_included_assets,
        'N_generations': N_generations,
        'max_time_sec': max_time_sec,
    })

    print('\t\tEC ok.')

```

Appendix 2. Results of the Wilcoxon signed-rank test for simulated data set

Appendix 2.1. Right-tailed Wilcoxon signed-rank test $H_0: U(GA) < U(MIQP)$

Parameters	GA average	MIQP average	P-value	Z-statistic
N = 500, K = 15, T = 10	0.017442	0.013313	0.000977	55.0
N = 500, K = 15, T = 15	0.016581	0.012912	0.000977	55.0
N = 500, K = 15, T = 30	0.017345	0.012360	0.000977	55.0
N = 500, K = 15, T = 60	0.016656	0.012171	0.000977	55.0
N = 500, K = 100, T = 10	-0.005556	-0.009924	0.000977	55.0
N = 500, K = 100, T = 15	-0.006970	-0.009928	0.000977	55.0
N = 500, K = 100, T = 30	-0.008807	-0.009929	0.000977	55.0
N = 500, K = 100, T = 60	-0.009605	-0.009932	0.000977	55.0
N = 500, K = 250, T = 10	-0.007881	-0.010064	0.000977	55.0
N = 500, K = 250, T = 15	-0.008472	-0.010064	0.000977	55.0
N = 500, K = 250, T = 30	-0.009213	-0.010064	0.000977	55.0
N = 500, K = 250, T = 60	-0.009834	-0.010064	0.000977	55.0

N denotes the number of investable assets, K denotes the maximum number of assets allowed in the portfolio, T denotes the time limit. The table has been created by the author.

Parameters	GA average	MIQP average	P-value	Z-statistic
N = 1000, K = 15, T = 15	0.019708	0.018228	0.009766	50.0
N = 1000, K = 15, T = 30	0.021060	0.016744	0.000977	55.0
N = 1000, K = 15, T = 60	0.020376	0.016270	0.000977	55.0
N = 1000, K = 100, T = 10	-0.004401	-0.010477	0.000977	55.0
N = 1000, K = 100, T = 15	-0.006180	-0.011327	0.000977	55.0
N = 1000, K = 100, T = 30	-0.009117	-0.011614	0.000977	55.0
N = 1000, K = 100, T = 60	-0.010681	-0.011793	0.000977	55.0
N = 1000, K = 250, T = 10	-0.006896	-0.012376	0.000977	55.0
N = 1000, K = 250, T = 15	-0.007555	-0.012376	0.000977	55.0
N = 1000, K = 250, T = 30	-0.009273	-0.012376	0.000977	55.0
N = 1000, K = 250, T = 60	-0.010764	-0.012376	0.000977	55.0
N = 1000, K = 15, T = 15	0.019708	0.018228	0.009766	50.0

N denotes the number of investable assets, K denotes the maximum number of assets allowed in the portfolio, T denotes the time limit. The table has been created by the author.

Parameters	GA average	MIQP average	P-value	Z-statistic
N = 1500, K = 15, T = 10	0.022520	0.031395	1.000000	0.0
N = 1500, K = 15, T = 15	0.022036	0.023468	0.903320	15.0
N = 1500, K = 15, T = 30	0.021615	0.019369	0.018555	48.0
N = 1500, K = 15, T = 60	0.021406	0.016501	0.000977	55.0
N = 1500, K = 100, T = 10	-0.004743	-0.010353	0.000977	55.0
N = 1500, K = 100, T = 15	-0.006270	-0.011038	0.000977	55.0
N = 1500, K = 100, T = 30	-0.009190	-0.012582	0.000977	55.0
N = 1500, K = 100, T = 60	-0.011646	-0.013285	0.000977	55.0
N = 1500, K = 250, T = 10	-0.007049	-0.013336	0.000977	55.0
N = 1500, K = 250, T = 15	-0.007837	-0.013794	0.000977	55.0
N = 1500, K = 250, T = 30	-0.009599	-0.014235	0.000977	55.0
N = 1500, K = 250, T = 60	-0.011590	-0.014322	0.000977	55.0

N denotes the number of investable assets, K denotes the maximum number of assets allowed in the portfolio, T denotes the time limit. The table has been created by the author.

Parameters	GA average	MIQP average	P-value	Z-statistic
N = 2000, K = 15, T = 10	0.024272	0.744491	1.000000	0.0
N = 2000, K = 15, T = 15	0.022777	0.387339	1.000000	0.0
N = 2000, K = 15, T = 30	0.022822	0.024218	0.934570	13.0
N = 2000, K = 15, T = 60	0.023522	0.018727	0.000977	55.0
N = 2000, K = 100, T = 10	-0.003985	-0.010148	0.000977	55.0
N = 2000, K = 100, T = 15	-0.006155	-0.010493	0.000977	55.0
N = 2000, K = 100, T = 30	-0.009250	-0.011676	0.000977	55.0
N = 2000, K = 100, T = 60	-0.011857	-0.013240	0.000977	55.0
N = 2000, K = 250, T = 10	-0.006960	-0.014015	0.000977	55.0
N = 2000, K = 250, T = 15	-0.007751	-0.014342	0.000977	55.0
N = 2000, K = 250, T = 30	-0.009449	-0.014905	0.000977	55.0
N = 2000, K = 250, T = 60	-0.011642	-0.015198	0.000977	55.0

N denotes the number of investable assets, K denotes the maximum number of assets allowed in the portfolio, T denotes the time limit. The table has been created by the author.

Parameters	GA average	MIQP average	P-value	Z-statistic
N = 2500, K = 15, T = 10	0.024090	0.747985	1.000000	0.0
N = 2500, K = 15, T = 15	0.022041	0.747985	1.000000	0.0
N = 2500, K = 15, T = 30	0.021477	0.031240	1.000000	0.0
N = 2500, K = 15, T = 60	0.021550	0.022838	0.838867	18.0
N = 2500, K = 100, T = 10	-0.005037	-0.011143	0.000977	55.0
N = 2500, K = 100, T = 15	-0.006781	-0.011321	0.000977	55.0
N = 2500, K = 100, T = 30	-0.010058	-0.012635	0.000977	55.0
N = 2500, K = 100, T = 60	-0.012710	-0.013724	0.000977	55.0
N = 2500, K = 250, T = 10	-0.007069	-0.015150	0.000977	55.0
N = 2500, K = 250, T = 15	-0.007831	-0.015471	0.000977	55.0
N = 2500, K = 250, T = 30	-0.010170	-0.016137	0.000977	55.0
N = 2500, K = 250, T = 60	-0.012508	-0.016557	0.000977	55.0

N denotes the number of investable assets, K denotes the maximum number of assets allowed in the portfolio, T denotes the time limit. The table has been created by the author.

Appendix 2.1. Left-tailed Wilcoxon signed-rank test $H_0: U(GA) > U(MIQP)$

Parameters	GA average	MIQP average	P-value	Z-statistic
N = 3000, K = 15, T = 10	0.024888	0.746543	0.000977	0.0
N = 3000, K = 15, T = 15	0.022455	0.746543	0.000977	0.0
N = 3000, K = 15, T = 30	0.022062	0.746543	0.000977	0.0
N = 3000, K = 15, T = 60	0.023124	0.027347	0.001953	1.0
N = 3000, K = 100, T = 10	-0.004018	0.742141	0.000977	0.0
N = 3000, K = 100, T = 15	-0.006126	-0.011206	1.000000	55.0
N = 3000, K = 100, T = 30	-0.009698	-0.012045	1.000000	55.0
N = 3000, K = 100, T = 60	-0.012764	-0.013447	0.998047	53.0
N = 3000, K = 250, T = 10	-0.006593	-0.015680	1.000000	55.0
N = 3000, K = 250, T = 15	-0.007383	-0.015743	1.000000	55.0
N = 3000, K = 250, T = 30	-0.009862	-0.016439	1.000000	55.0
N = 3000, K = 250, T = 60	-0.012496	-0.016855	1.000000	55.0

N denotes the number of investable assets, K denotes the maximum number of assets allowed in the portfolio, T denotes the time limit. The table has been created by the author.

Parameters	GA average	MIQP average	P-value	Z-statistic
N = 3500, K = 15, T = 10	0.024931	0.751761	0.000977	0.0
N = 3500, K = 15, T = 15	0.023897	0.751761	0.000977	0.0
N = 3500, K = 15, T = 30	0.022954	0.751761	0.000977	0.0
N = 3500, K = 15, T = 60	0.022448	0.032181	0.000977	0.0
N = 3500, K = 100, T = 10	-0.004121	0.748370	0.000977	0.0
N = 3500, K = 100, T = 15	-0.006469	0.672645	0.001953	1.0
N = 3500, K = 100, T = 30	-0.009678	-0.011317	1.000000	55.0
N = 3500, K = 100, T = 60	-0.012683	-0.012977	0.883789	39.0
N = 3500, K = 250, T = 10	-0.006787	0.746450	0.000977	0.0
N = 3500, K = 250, T = 15	-0.007869	-0.014222	1.000000	55.0
N = 3500, K = 250, T = 30	-0.009885	-0.014955	1.000000	55.0
N = 3500, K = 250, T = 60	-0.012751	-0.016297	1.000000	55.0

N denotes the number of investable assets, K denotes the maximum number of assets allowed in the portfolio, T denotes the time limit. The table has been created by the author.

Parameters	GA average	MIQP average	P-value	Z-statistic
N = 4000, K = 15, T = 10	0.025951	0.747255	0.000977	0.0
N = 4000, K = 15, T = 15	0.024223	0.747255	0.000977	0.0
N = 4000, K = 15, T = 30	0.024162	0.747255	0.000977	0.0
N = 4000, K = 15, T = 60	0.023014	0.747255	0.000977	0.0
N = 4000, K = 100, T = 10	-0.003982	0.747255	0.000977	0.0
N = 4000, K = 100, T = 15	-0.006206	0.747255	0.000977	0.0
N = 4000, K = 100, T = 30	-0.009197	-0.012074	1.000000	55.0
N = 4000, K = 100, T = 60	-0.012190	-0.013074	0.998047	53.0
N = 4000, K = 250, T = 10	-0.006557	0.740857	0.000977	0.0
N = 4000, K = 250, T = 15	-0.007375	0.364383	0.116211	15.0
N = 4000, K = 250, T = 30	-0.009708	-0.015017	1.000000	55.0
N = 4000, K = 250, T = 60	-0.012201	-0.015940	1.000000	55.0

N denotes the number of investable assets, K denotes the maximum number of assets allowed in the portfolio, T denotes the time limit. The table has been created by the author.

Parameters	GA average	MIQP average	P-value	Z-statistic
N = 4500, K = 15, T = 10	0.025742	0.750432	0.000977	0.0
N = 4500, K = 15, T = 15	0.024286	0.750432	0.000977	0.0
N = 4500, K = 15, T = 30	0.022360	0.750432	0.000977	0.0
N = 4500, K = 15, T = 60	0.023244	0.750432	0.000977	0.0
N = 4500, K = 100, T = 10	-0.003755	0.748184	0.000977	0.0
N = 4500, K = 100, T = 15	-0.005543	0.750432	0.000977	0.0
N = 4500, K = 100, T = 30	-0.009357	-0.012708	1.000000	55.0
N = 4500, K = 100, T = 60	-0.012405	-0.013590	0.997070	52.0
N = 4500, K = 250, T = 10	-0.006590	0.747506	0.000977	0.0
N = 4500, K = 250, T = 15	-0.007559	0.516201	0.013672	6.0
N = 4500, K = 250, T = 30	-0.009647	-0.015837	1.000000	55.0
N = 4500, K = 250, T = 60	-0.012626	-0.016489	1.000000	55.0

N denotes the number of investable assets, K denotes the maximum number of assets allowed in the portfolio, T denotes the time limit. The table has been created by the author.

Parameters	GA average	MIQP average	P-value	Z-statistic
N = 5000, K = 15, T = 10	0.026617	0.747252	0.000977	0.0
N = 5000, K = 15, T = 15	0.024789	0.747252	0.000977	0.0
N = 5000, K = 15, T = 30	0.022884	0.747252	0.000977	0.0
N = 5000, K = 15, T = 60	0.023673	0.747252	0.000977	0.0
N = 5000, K = 100, T = 10	-0.003494	0.747252	0.000977	0.0
N = 5000, K = 100, T = 15	-0.006076	0.747252	0.000977	0.0
N = 5000, K = 100, T = 30	-0.009214	0.064842	0.967773	45.0
N = 5000, K = 100, T = 60	-0.012671	-0.013446	0.995117	51.0
N = 5000, K = 250, T = 10	-0.006550	0.744392	0.000977	0.0
N = 5000, K = 250, T = 15	-0.007356	0.747252	0.000977	0.0
N = 5000, K = 250, T = 30	-0.009700	-0.016411	1.000000	55.0
N = 5000, K = 250, T = 60	-0.012550	-0.017255	1.000000	55.0

N denotes the number of investable assets, K denotes the maximum number of assets allowed in the portfolio, T denotes the time limit. The table has been created by the author.