Department of Computing Science
Umeå University

May 24, 2023

Degree Project in Computing Science Engineering, Spring 2023

Minimizing initial margin requirements using computational optimization

Jacob Ahlman Bohm
jacobab@cs.umu.se

Course responsible
Henrik Björklund

Supervisors

Internal Patrik Eklund
External Jacob Titus

Abstract

Trading contracts with future commitments requires posting a collateral, called initial margin requirement, to cover associated risks. Differences in estimating those risks and varying risk appetites can however lead to identical contracts having different initial margin requirements at different market places. This creates a potential for minimizing those requirements by reallocating contracts.

The task of minimizing the requirement is identified as a black-box optimization problem with constraints. The aim of this project was to investigate that optimization problem, how it can best be tackled, and comparing different techniques for doing so. Based on the results and obstacles encountered along the way, some guidelines are then outlined to provide assistance for whomever is interested in solving this or similar problems.

The project consisted both of a literature study to examine existing knowledge within the subject of optimization, and an implementation phase to empirically test how well that knowledge can be put to use in this case. During the latter various algorithms were tested in a number of different scenarios. Focus was put on practical aspects that could be important in a real situation, such as how much they could decrease the initial margin requirement, execution time, and ease of implementation.

As part of the literature study, three algorithms were found which were evaluated further: simulated annealing, differential evolution, and particle swarm optimization. They all work without prior knowledge of the function to be optimized, and are thus suitable for black-box optimization.

Results from the implementation part showed largely similar performance between all three algorithms, indicating that other aspects such as ease of implementation or parallelization potential can be more important to consider when choosing which one to use. They were all well able to optimize different portfolios in a number of different cases. However, in more complex situations they required much more time to do so, showing a potential need to speed up the process.

Acknowledgements

I would like to thank both my supervisors, Patrik Eklund and Jacob Titus, for the support and feedback I have received during this project - it has been much appreciated. I would also like to thank Eduardo Paludo Gomes at Nasdaq for providing valuable input and helped explaining the financial parts that can be quite confusing for me as a computer scientist.

Contents

1 Introduction **1**
1.1 Problem formulation 1

2 Background **2**
2.1 Financial background 2
2.1.1 Financial instruments 2
2.1.2 Clearing house 3
2.1.3 Initial margin 4
2.2 Defining the optimization problem 5
2.3 Dimensionality of the objective function 6
2.4 Categorization of optimization strategies 7
2.5 Related work 7

3 Method **8**
3.1 Evaluation criteria 8
3.2 Literature study 9
3.3 Implementation phase 9
3.3.1 Constructing the objective function 10
3.3.2 Handling constraints 11
3.3.3 Finding optimal parameters for the algorithms 12
3.4 Test design 12
3.4.1 Test data and scenarios 13

4 Optimization algorithms **14**
4.1 Simulated annealing 14
4.2 Differential evolution 15
4.2.1 Mutation schemes and parameters 17
4.3 Particle swarm optimization 18
4.3.1 Population size and other parameters 18
4.4 Other algorithms 20

5 Results **20**
5.1 Finding the solution in simple scenarios 20
5.2 Scaling up the problem 22
5.3 Execution time 23

6 Discussion **26**
6.1 Limitations 26
6.2 Potential for optimizing initial margin requirements 27
6.3 Choosing which algorithm to use 28

7 Conclusion and future work **29**

8 References **30**

A Algorithm details and parameter values **32**
A.1 Simulated annealing 32
A.2 Differential evolution 32
A.3 Particle swarm optimization 33

B Full results **34**
B.1 Portfolio C 34
B.2 Portfolio D 36
B.3 Portfolio E 38

List of Tables

1 Common differential evolution mutation schemes 17
2 Number of times the optimal allocation is found by each optimizer 22

List of Figures

1 Visualization of bilateral trades between brokers and trading with a clearing house 3
2 Visualization of the differences between absolute and relative portfolio allocation 11
3 Simulated annealing in action 15
4 Differential evolution in action 17
5 Particle swarm optimization in action 19
6 Portfolio A optimization process using different algorithms 21
7 Process of optimizing portfolio B using simulated annealing 21
8 Portfolio C optimization results 22
9 Portfolio D optimization results 23
10 Portfolio E optimization results 23
11 Timing of the different parts of the optimization process 24
12 Execution time per evaluation when optimizing portfolio C 25
13 Execution time per evaluation when optimizing portfolio D 25
14 Execution time per evaluation when optimizing portfolio E 25
15 Portfolio C results, 200 evaluations 34
16 Portfolio C results, 400 evaluations 34
17 Portfolio C results, 800 evaluations 35
18 Portfolio C results, 1,600 evaluations 35
19 Portfolio D results, 200 evaluations 36
20 Portfolio D results, 400 evaluations 36
21 Portfolio D results, 800 evaluations 37
22 Portfolio D results, 1,600 evaluations 37
23 Portfolio E results, 200 evaluations 38
24 Portfolio E results, 400 evaluations 38
25 Portfolio E results, 800 evaluations 39
26 Portfolio E results, 1,600 evaluations 39

List of Algorithms

1 Simulated annealing 14
2 Differential evolution (DE/rand/1) 16
3 Particle swarm optimization 19

1 Introduction

There exist many different types of financial instruments traded at various market places, many of which come with future obligations. One such instrument is the futures contract, which is an agreement to buy something in the future. As there is always the risk of any of the involved parties not being able to deliver on their part of the contract, exchanges where futures and other similar instruments are traded require a collateral. This collateral is called the margin requirement, a part of which is the initial margin (IM).

Different methods exist to calculate the IM requirement, and different market operators may also have varying risk appetites, which means that the same set of instruments and associated positions may have different IM requirements at different market places. This creates opportunities for traders that want to minimize their margin requirements, without changing their portfolio exposure. This opportunity can be described as an optimization problem: How should the instruments be allocated among the different available markets, in order to achieve an initial margin requirement that is as low as possible?

There are, however, several complicating factors that leads to the mentioned optimization problem being non-trivial to solve. The initial margin requirement needs to be calculated on a whole portfolio and not as a sum of individual IM requirements for each product, and the varying and often complex methods used for calculating the requirement means that it is very difficult to use information about the function itself to alleviate the optimization.

It should also be mentioned is that while theoretical knowledge can provide good and clean answers to many problems, and there has certainly been a vast amount of research conducted in the field of optimization, reality often has a way of complicating things. When attempting to apply the theory in a real situation, more often than not there exist many unique challenges that need to be addressed.

This thesis was conducted in collaboration with Nasdaq, investigating how the optimization problem could best be tackled. While doing this, focus was being put on addressing both the unique set of challenges associated with the specific optimization problem itself, and the obstacles of applying theoretical knowledge in a practical situation.

1.1 Problem formulation

The aim of this project is to investigate the feasibility of applying computational optimization for the particular use case of minimizing initial margin requirements. While building on a foundation of existing theoretical knowledge, the focus of the project lies in identifying the unique obstacles encountered when putting that knowledge to use. The first objective is to determine whether it is even practical to apply optimization in this scenario. If it is, the secondary objective is to find information about the challenges that exist in doing so, in order to provide guidelines for how they can be addressed.

As such, the purpose of this project will not necessarily be to implement the best possible optimizer for this situation. Rather, the project can be regarded as a prestudy for such an optimizer: By breaking down the problem and looking at potential solutions, guidance

can be provided for someone faced with solving it. Also, while the primary focus is to provide assistance for solving this specific optimization problem, hopefully the project can provide some guidance for solving other but similar problems as well.

More specifically, the project attempts to answer the following questions:

- What are the characteristics of this optimization problem, and which techniques exist for solving problems with those characteristics?
- How well do the techniques perform in terms of speed, accuracy, capabilities and ease of implementation?
- What are the practical difficulties associated with solving this particular optimization problem, and how can they be addressed?
- Based on the information gathered, is it practical to apply computational optimization to solve this problem? If so, what would be a good approach forward?

2 Background

This section covers some background knowledge that is required in order to fully understand the problem. It both describes some central financial terms and provides a more precise definition of the optimization problem that is central for the thesis.

2.1 Financial background

This report focuses on the financial instrument known as futures, and the initial margin related to trading that instrument. A futures contract is a kind of derivative, meaning that its value is derived from some kind of underlying asset [1, p. 23]. While this project focuses solely on futures, other derivatives such as options for example also work in a similar manner. Therefore much of the theory and results may be applicable to those as well.

2.1.1 Financial instruments

A futures contract is a normally exchange-traded agreement to either buy or sell a certain asset for a predetermined price at some specific date in the future. The trader can take one of two positions with a futures contract: A long position for buying the underlying asset, or a short position for selling it [1, p. 30].

This report mentions things such as a given instrument being available at several different markets, but it should be noted that this is a simplification of the reality for practical reasons. What is actually meant by this is that there exist instruments at those markets that are regarded as *equivalent*, i.e. they may be derived from the same underlying commodity in a similar fashion, but they are still technically different contracts. A gold

future at one market may for example be exchanged for another, but for all practical purposes equivalent, gold future at another clearing market. In this thesis, "instrument pairing" is used to denote two or more interchangeable contracts at different markets.

In the cases investigated as part of this thesis there are virtually be no difference between the different instruments of a pairing, save for the fact that they may have different sizes and as such may need a multiplicative factor when converting between them. However, this may not always be the case in the real world. Especially with more complicated or obscure contracts than those investigated here, it may be difficult or even impossible to find true equivalents at other markets, and in such cases the differences between them must be more carefully considered if they are to be paired.

2.1.2 Clearing house

Futures and other contracts are in general traded with the help of an intermediary called a "clearing house". Trading at a clearing house can only be done by members of the clearing house, or with the help of one of its members. The clearing house acts as a counter-party during a trade, meaning that the members only need to do business with a single entity instead of having several bilateral agreements [1, p. 53-54], as visualized by figure 1.

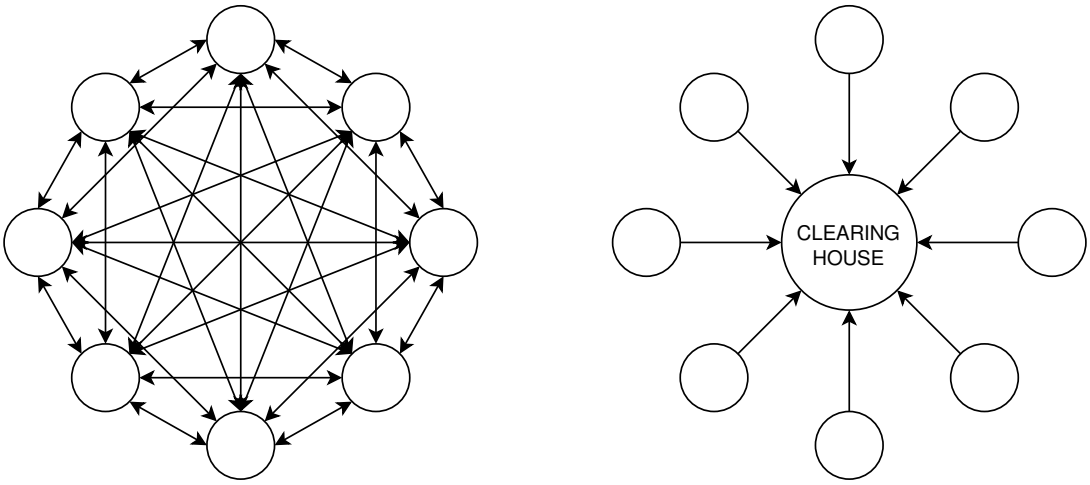


Figure 1: Visualization of bilateral trades between brokers (left) and trading with a clearing house as counterparty (right)

Since the clearing house is acting as the counter-party during the trade of the contracts, this also means that it takes on some of the risk associated with it. In particular it needs to be able to cover the losses in case one of the members of the house defaults. In order to do this, the house needs to have a guaranty fund. Each member will have to contribute to this fund, with the amount depending on the size and risk of their positions.

This guaranty fund contribution is what constitutes the margin requirement mentioned in the introduction of the report. While there exist several components of the margin requirements, some of which are related to changes in contract price, this thesis focuses solely on the initial margin component - which as the name suggests is the initial collateral required for a given portfolio of contracts [1, p. 53-54].

2.1.3 Initial margin

Different clearing houses often use different methodologies to calculate the initial margin (IM) requirement. Most methods fall into one of two different categories: Standard Portfolio Analysis of Risk (SPAN) or Value-at-Risk (VaR). There are, however, many variations of both methods, and together with differences in parameter values and risk appetite between clearing houses this can result in different margin requirements at different houses for what is essentially the same portfolio.

Standard Portfolio Analysis of Risk based methods use market simulations to try to estimate the maximum likely loss. The core of how the method works is to first group instruments together by their underlying asset, and then look at a number of scenarios (often 16) simulating changes in volatility and price of the underlying asset. The largest loss from all scenarios, called the Scan Risk, is then used as a base for calculating the initial margin [2].

Value-at-Risk based methods attempts to estimate the maximum possible loss in a given time span with a certain confidence level. For example, a VaR method could estimate something along the lines of *"With 99% certainty the portfolio will at most lose X USD in the coming N days"*. The two most common ways of estimating this is to either use historical data, or using a model-based method (which in turn use parameters derived from historical data) [1, p. 514].

All these methods, both SPAN and VaR based ones, attempt to predict the future based on historical data in some way or another. This is something which is inherently impossible to do perfectly. As they differ in how they do this, depending on the method used the estimated risk and in extension the initial margin requirement may differ for the same portfolio of contracts. It may also be the case that different clearing houses have different risk appetites, and as such may require different initial margin requirements for the same estimated risk.

The different methods used and various risk appetites is part of what creates an opportunity for optimization, but there are also more factors at play. Another major factor is that contracts can offset each-other, and as such reduce the perceived risk and therefore also IM requirement, but that this offset is only possible when those contracts exist at the same clearing house.

As an example of this, a customer may be interesting in trading a gold futures contract. They are interested in having both a long position of 100 and a short position of 50. If both these positions are held at the same clearing house, they will offset each-other as they are opposing positions. The risk will then be calculated based on the net 50 long position. If the customer would instead place the 100 long position and 50 short position at different clearing houses, they would not be netted against each-other and instead the risk will be calculated for each of those two parts individually, resulting in a much higher IM requirement.

It should be stated that the above-mentioned example is a very simple scenario, in real situations may be much more complicated and as such the "correct" portfolio allocation may not be so trivial. Different contracts may still be partially netted against each-other if their values are correlated. For example gold and copper are two different assets, each

having different market prices, but the prices may very well be correlated to a certain degree. As such, their futures contract may still somewhat offset one another, resulting in a lower calculated risk and IM requirement. This is called the *inter-commodity spread risk* [2].

Exploiting this potential for lower initial margin requirements by having offsetting at the same clearing houses is easy for very simple portfolios, but as the portfolios grow larger and more complex instruments are used it becomes increasingly harder to do manually. It is however yet another factor that creates potential for lowering the IM requirement, creating many opportunities for an optimizer that is able to effectively find and exploit this knowledge.

2.2 Defining the optimization problem

More formally, we have a set of n instruments $P = \{P_1, P_2, ..., P_n\}$ that we can distribute among m portfolios $PF = \{PF_1, PF_2, ..., PF_m\}$ at different clearing houses. Each available instrument-portfolio pair (note that not all instruments are necessarily available at all clearing houses) has an associated net position. The position in portfolio i featuring instrument j will be denoted as $PP_{i,j} \in \mathbb{R}$. This position can be zero, indicating that the instrument is not present in that portfolio.

Given the m portfolios, each with their own method of calculating the IM requirement, the IM requirement for a single portfolio $1 \leq i \leq m$ can be described as

$$IM_i = MM_i(PP_i) \ , \tag{1}$$

where MM_i is the margin model used to calculate the IM for that portfolio, PP_i is the set of instrument positions in that portfolio and IM_i is the resulting initial margin for portfolio i. The total initial margin can then be described as

$$IM_{total} = \sum_{i}^{m} IM_i \ , \tag{2}$$

which we want to minimize. In other words, we want to find the optimal portfolio combination s.t. we have $\min(IM_{total})$.

Note that the initial margin must be calculated on a per-portfolio basis. It is not possible to calculate the initial margins for all instruments in the portfolio individually and merely sum them together, since the instruments affect each-other. For example, a portfolio with both a long and short position of the same instrument and expiry date will have a lower risk since they more or less cancel each-other out, thus the IM requirement for the portfolio as a whole will be lower than the sum of the requirements of its individual parts. As such, the optimization problem is non-separable.

The objective function IM_{total} is very complex and is also subject to change as the margin model or its parameters can change, so for all practical intents and purposes

it can essentially be regarded as unknown. This type of optimization problem where the function to be optimized is not known is commonly called *black-box optimization*, or *derivative-free optimization* since we cannot use any derivative of the objective. Since no assumptions are made about the objective function it must be regarded as non-convex, meaning that a local minimum is not guaranteed to be a global minimum.

The problem also has constraints since the portfolio exposure, the values of its positions, should remain constant - otherwise the optimal solution would always be to simply sell all instruments since an empty portfolio is risk-free. The function is also limited in such a way that no offsetting positions are allowed. That is, a net long position for example can only be replaced with other net long positions. It cannot be increased by introducing a net short position at another clearing house, and this creates another constraint that needs to be handled.

To summarize, minimizing the objective function is in this case a non-convex, non-separable black-box optimization problem with constraints.

2.3 Dimensionality of the objective function

As discussed in the previous section, the objective function consists of distributing n instruments continuously among m portfolios. This would indicate that the dimensionality D of the objective function $f : \mathbb{R}^D \to \mathbb{R}$ is $D = n \times m$, where n is the number of instruments and m the number of clearing houses. However, the reality is most often not that simple. While it is true that the formula provides an upper limit for the dimensionality of the function (i.e. $D \leq n \times m$), it is often lower than that due to the fact that not all instruments are available at all markets. Rather, a given instrument is commonly only available at a subset of the available markets.

An effect of this fact is that just knowing the number of dimensions D may not provide as much information about the objective function as it could have done. This is because that it is possible that the nature of the function depends a lot on the underlying structure of the problem, and given the above-mentioned situation that structure is poorly reflected by the dimensionality alone.

To provide a concrete example of this, consider two situations:

(A) Ten instruments which are to be distributed among two portfolios, with all instruments being available at all portfolios.

(B) Ten instruments which are to be distributed among 20 portfolios, but with each portfolio only supporting a single instrument (meaning that each instrument can be distributed among two portfolios, and no portfolio containing more than one kind of instrument).

Situation A and B will both result in an objective function with 20 dimensions, but given how initial margin calculations are done the nature of those functions could be very different. Following from that, the prospects of finding a good optimization, as well as the best way to do it, may differ a lot. This example is also just one of many. There are

countless possible structures of the underlying problem, which may still have the same dimensionality.

As a result, this raises the need for diverse test data to provide a decent indication of how the optimizers perform in practice. While it will always be impossible to capture all possible situations, it can be good to at least have not only a number of different-dimensionality objective functions, but also cover a range of different instrument-portfolio combinations.

2.4 Categorization of optimization strategies

There have been extensive work done in the field of optimization, which has lead to numerous different techniques available. This creates a need to categorize them into different groups in order to get an overview of and be able to compare the different techniques.

One of the more general distinctions that can be made is categorizing algorithms as either derivative-based or derivative-free methods [3, p. 5]. The former uses the derivative of the objective function (or an approximation of it) in some way in order to improve the process, which can result in very good results, but they are also very limited in terms of which problems they can be applied to. The latter works regardless of any information about the objective function or its derivative, which make it much more general at the cost of performance. Since the objective function studied in this paper is regarded as an unknown black-box function, we are limited to the use of derivative-free algorithms and as such the study will focus on those.

There are also many other general characteristics which can be used to further differentiate between and categorize different optimization algorithms. They can be either deterministic or stochastic, where a deterministic algorithm will always result in the same result given the exact same precondition, whereas stochastic algorithms include at least some degree of randomness [3, p. 5]. Furthermore there are both trajectory-based and population-based methods, where the former use a single agent and the latter use multiple agents for traversing the search space.

The algorithms used in this study are discussed in more detail in section 4, where these categorization terms are used to signify the similarities and differences between the different algorithms.

2.5 Related work

Countless studies have been conducted in the area of computational optimization, and also specifically the sub-area of black-box optimization. Although these often feature some of the very same algorithms that will be featured in this project, many of these focus on the algorithms themselves [4]–[6], while this project will put emphasis on the practical application of the algorithms. As such, insights from those studies will be used as a foundation from which the practical problem this project focuses on can be solved.

Certainly there also exists other research that focuses on the application of one or several optimization algorithms, but something that should be mentioned with regards to this is the so-called "no free lunch theorem" [7], which essentially states that no single algorithm can be best for all optimization problems. Rather, if a given algorithm is well-suited for a specific problem, this needs to be offset by it being ill-suited for another kind of optimization problem.

The implications of this theorem is that all situations are unique, and that one algorithm having shown good results in one situation does not necessarily mean that it will do so in another. As such, even if the practical application of an algorithm has been investigated in one situation, this research cannot be directly inferred to another situation. This combined with the fact that no existing research has been found that focuses on the specific task of IM optimization, there is an interest to investigate the topic and its unique challenges.

3 Method

The thesis project consisted of two phases which are described in this section. The first was a literature study aimed at identifying optimization techniques which were thought to be relevant for this kind of problem. The second phase was the implementation phase, where the most promising methods identified in the literature study were implemented in order to test how well they performed in practice.

3.1 Evaluation criteria

Both the literature study and implementation phase aimed to filter out and analyse different optimization algorithms, in order to find one that is suitable for the problem at hand. The identified algorithms were evaluated based on a number of evaluation criteria, which can be summarized with the following general criteria:

- How well the algorithm can minimize the objective function.
- Execution time required, putting an emphasis on algorithms that are able to receive adequate results in a reasonable time-frame.
- Ease of implementation, the algorithm should not be difficult for the programmer to understand and apply.

The listed criteria aimed to capture properties that are often important in a real situation. While often not necessary to find the *solution* of the optimization problem in a strict sense, i.e. the global optimum of the objective function, the user often wants a result that is "good enough". Although finding the best possible portfolio allocation would obviously be good, this needs to be weighted against the cost of searching for it and the desire to have the result in a reasonable amount of time. Taking those things into account it is often better to have a decent result fast than waiting for the perfect result.

In the project no hard limits were used for its evaluation criteria, as those rarely reflect what is actually useful in a real situation. For example, there is no binary "cut-off" in terms of execution time when the optimizer stops being useful, but of course it is beneficial to make it as fast as possible. What is a reasonable amount of time is therefore not strictly defined, rather being something that depends on the specific use case. However, as a general guideline, an execution time of at most a couple of seconds would probably be fast enough for most cases. Likewise, what is good enough performance and easy enough to implement is also up to the reader to decide, this study merely attempts to provide insights and guidelines to help with making an informed decision.

For each of the two phases of the project, the general criteria were broken down into more specific criteria that were used to determine exactly what would be evaluated during that phase.

3.2 Literature study

The first part of the project was a literature study, the purpose of which was two-fold. It both attempted to identify and get an overview of the available algorithms, and filter among them in order to select a few promising candidate methods. These candidates were then subject to further evaluation during the implementation phase.

The filtering was done by examining existing research and determining whether a given algorithm is suitable based on

- whether the algorithm works without any previous knowledge of the objective function or its derivative
- how well the algorithm handles non-convex objective functions, and whether it is capable of escaping or avoiding the traps of local minima
- the complexity of the algorithm and whether it can be easily implemented for the given problem.

The goal of those criteria was to select a few algorithms that were deemed effective and suitable for the problem, while at the same time putting focus on practicalities such as reducing development time. Another reason to strive for simple solutions to the degree it is possible, is that it makes it easier to maintain the code and expand functionality.

3.3 Implementation phase

Theory does not take into account all the intricacies of reality, and as such it was not possible to determine whether an algorithm is suitable for a given situation using literature alone. The purpose of the implementation phase was to empirically test how well the candidate algorithms identified during the literature study performed in practice.

Once all algorithms were implemented, they were tested in a number of different scenarios (see section 3.4.1), using the relative change in initial margin as primary measurement of

performance. In other words, the tests provided data which was examined to determine how well the objective function could be minimized using the different algorithms in different scenarios. More specifically, the scenarios that were examined were

- whether the implementations are able to find the global optima in a simple scenario
- how well the different algorithms scale with an increase in portfolios and/or instruments, i.e. an increase in dimensionality of the objective function.

3.3.1 Constructing the objective function

While in theory constructing the objective function is trivial - simply choose how much of a given instrument should be allocated to each portfolio - in practice there exist some complicating factors and design choices that could affect the performance. This section describes some general technical details of how the function was implemented.

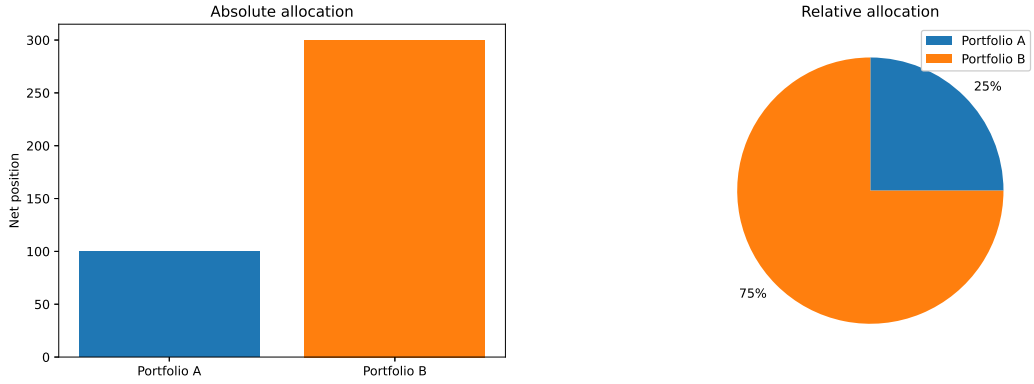
With regards to how the actual portfolio allocations are handled there exist primarily two alternatives, absolute and relative:

- **Absolute allocation** means the input to the objective function will directly decide the positions of the different instruments in the different portfolios. For example, the input could be that portfolio A should have a net short position of 100 of a specific futures contract, and portfolio B a net position of 300 for the equivalent contract.
- **Relative allocation**, on the other hand, only allows to set the relative weights of the different portfolios. This could be done by allocating 25% of a short position for a given futures contract to portfolio A, and 75% of the position to portfolio B.

Both of the mentioned examples would result in the same portfolios and as such also the same resulting initial margin requirement (for the purpose of the example it is assumed that the contract sizes and their prices are equal at both markets). The practical difference between the two instead lie in the fact that an absolute allocation gives the caller the ability to change the total size of the positions, whereas a relative allocation does not. For example, given an absolute allocation it would be possible to take a net position of 200 in portfolio A and 600 in portfolio B, something which would change the total exposure. This is not allowed given the problem description and as such it would put a lot of responsibility on the optimizer to prevent this.

The difference between the two allocation strategies is exemplified by figure 2, where with the absolute allocation the optimizer can change the size of the bars, but with the relative allocation it is only possible to change the proportion of the pie slices - not the size of the pie itself.

The objective function used during this project used a **relative allocation** as its input. This was chosen to reduce the need for constraint handling, thanks to exposure being kept constant without needing to be explicitly accounted for. Translating the relative



(a) Absolute allocation (b) Relative allocation

Figure 2: Visualization of the differences between absolute and relative portfolio allocation

weights to actual positions and accounting for things such as different contract sizes are done implicitly by the objective function.

As a performance-increasing measure, caching was also added to all optimizers. Every time a new point in the search space is evaluated by an optimizer, that point and the resulting value is added to a cache. If the optimizer then attempts to evaluate that point again later on during its lifetime, the cached value will be fetched instead, thus eliminating the need for re-evaluating that point. In the tests where a set number of objective function evaluations was the limiting factor for the optimizers, cache fetches did *not* count towards that limit - only new evaluations did.

3.3.2 Handling constraints

By using relative allocation of instruments at different markets the need for handling constraints is reduced, but not eliminated.

One problem that can still arise is that the total weight of all allocations for a single instrument could theoretically be zero. While naturally it needs to be possible to set the weight of a single or several instrument-portfolio allocation pairs to zero, meaning that no positions of that instrument will be present in that or those portfolios, this cannot be true for all portfolios at the same time for that instrument. In other words, each instrument must be allocated to at least one portfolio in order to keep the exposure constant.

Another problem is that the weights need to be kept within certain boundaries. More specifically, no weight should be negative as offsetting positions are not to be used (as explained in section 2.2). It should be noted, however, that if for some other use case offsetting positions would be allowed, this constraint could be removed. A possible middle-ground would also be to relax it to a soft constraint rather than a hard constraint, which would allow offsetting positions only when there is sufficient gain.

Handling constraints when optimizing can be done in many ways which all have their pros and cons. In this study two simple but popular methods were be combined in order

to handle the constraints: clamping and penalty terms. The clamping will ensure all values are within the allowed search space, for example to ensure no negative weights are used, and the penalty term is used to penalize any unwanted or illegal allocation that may occur within the search space.

3.3.3 Finding optimal parameters for the algorithms

Many optimization algorithms make use of different parameters that affect their performance and allow for fine-tuning their behaviour. In most cases these are simple numerical parameters, such as the population size in the case of population-based methods or values determining how fast the algorithm will converge and switch from global to local search. However, in some cases there also exist choices for controlling the more general strategy of a method. The different differential evolution schemes constitute one example of this (see section 4.2.1). Research shows that which values are chosen for these parameters can possibly have a major impact on how well an algorithm performs [8].

One thing to keep in mind when choosing the correct parameters is that just as when choosing algorithm, there is no silver bullet solution. No set of parameters will always perform best in all situations, as exemplified by the no free lunch theorem discussed in section 2.5. Rather, there are many factors that will affect which parameters work best in a given situation. Two examples of such factors are the nature of the function to be optimized, and how many objective function evaluations the optimizer is allowed to make.

Attempting to find the the optimal parameters for a given type of problem is a whole research topic on its own. In fact, there have been several studies using different tools to do just that, including the application of neural networks to optimize differential evolution parameters [9]. Due to the immense work associated with attempting to identify the optimal parameters, this paper does *not* focus on that. Instead, attempts were made to identify some "good enough" parameters for each algorithm based on a combination of existing research as well as some limited testing. The focus was not to provide the best possible performance, but rather to ensure that each algorithm has parameters allowing at least decent performance. This was done in order to increase the quality of comparisons between the different methods.

3.4 Test design

The implemented algorithms were tested in a number of scenarios. The primary measurement of performance was the average relative reduction in IM requirement, i.e. how well the objective function could be minimized. Secondary aspects that were measured were execution time and consistency. All tests were repeated 100 times unless otherwise noted.

All tests were performed on a MacBook Pro featuring an Intel i9-9880H CPU @ 2.30 GHz and 32 GB RAM. For all tests where the execution time was relevant, care was taken to reduce the risk of other applications running on the computer hogging resources as it could skew the data. Those tests were also never run concurrently, but rather one at a time.

3.4.1 Test data and scenarios

Underlying information such as the instruments themselves and other information relevant for the IM calculations are based on real, historic data. The portfolio constitutions however were artificially constructed as there were no real data available. The portfolios were constructed either manually or semi-randomly with the intent to create situations where there is potential for optimization. In some cases noise was added to make the situation more challenging for the optimizer algorithms, and also to be able to identify how well they scale as the problem size increases.

The following portfolios were constructed:

(A) Consists of one instrument (fuel oil futures) that is available at two markets, as well as another offsetting instrument position (futures spread) statically placed at one of those markets. This is a trivial situation where there should be great potential for optimization simply by having the two instruments at the same market. $D = 1$.

(B) Similar to portfolio A, this portfolio also features two correlated instruments (gold and silver futures contracts) with offsetting positions available at two markets. There are however two differences to portfolio A: The instruments are different, and they can both be swapped between the markets. $D = 2$.

(C) Based on portfolio B, but with several more or less unrelated instruments added to provide noise. This puts pressure on the optimizers' ability to navigate through that noise and still be able to find the optimization potential. $D = 40$.

(D) A static portfolio containing a number of different futures using various metals and energy products as underlying assets, spread across three clearing houses. The idea was to create a more complex situation where there are several different instruments with many correlations, meaning that the optimal solution is not trivial to find. $D = 80$.

(E) A large, randomized portfolio. The primary purpose of this one is to test the scalability of the optimizers, testing their ability to find the solution in reasonable time even in more complex situations. While the portfolio will always use the same instruments, their positions were randomized for each test run with this portfolio, so a larger variance in the outcome was expected. $D = 222$.

While the different portfolios attempt to capture a number of different situations that can be interesting to look at, the construction of them was somewhat constrained by limitations in the underlying data. A concrete example of this is that it could be interesting to investigate how the optimizers perform when there are many different available markets to choose between, but alas there are few instruments that have good pairings between more than two markets.

The tests conducted with the portfolios focused on how well the algorithms were able to minimize the objective function and the execution time required to do so given different number of objective function evaluations. They were designed with the aim to both find out how well the implementations can tackle the challenge, and if there are any differences between the algorithms when doing so.

4 Optimization algorithms

This section describes the optimization techniques that were identified as part of the literature study which were deemed promising according to the evaluation criteria. The section aims to provide an overview of these candidate algorithms and the reasoning behind choosing them for further evaluation during the implementation step. Featured here are also short descriptions of the parameters each algorithm require and how their values were chosen. The actual parameter values used are however not here, instead an overview of them can be found in appendix A.

Of the three algorithms chosen for further evaluation, two are population-based and one is trajectory-based. One thing that is not investigated further in this report, but should still be mentioned is that population-based algorithms are generally much easier to parallelize. Each individual of the population can in theory be evaluated concurrently, whereas in most trajectory-based methods the evaluation of each new iteration is dependent on the result from the previous one.

4.1 Simulated annealing

Simulated annealing (SA) is a well-established optimization algorithm, being developed by Kirkpatrick, Gelatt, and Vecchi as one of the first meta-heuristic algorithms back in 1983 [10]. It is an extension of the traditional hill climbing, with an added a degree of randomness that lets it escape local minima. This is an important feature for this problem, as it allows the algorithm perform well for objective functions with several local optima.

Algorithm 1 Simulated annealing

$s \leftarrow s_0$ ▷ State
$T \leftarrow T_0$ ▷ Temperature
$evaluations \leftarrow 0$
while $evaluations < maxEvaluation$ **do**
  $s_{new} \leftarrow$ random neighbour of s
  $Change \leftarrow \text{EVAL}(s_{new}) - \text{EVAL}(s)$
  if $Change \leq 0$ **then**
    $s \leftarrow s_{new}$
  else if $\text{RANDOM}(0, 1) \leq e^{-Change/T}$ **then**
    $s \leftarrow s_{new}$
  end if
  $T \leftarrow newTemperature(T)$
  $evaluations \leftarrow evaluations + 1$
end while
return s

There are many variations of the SA algorithm, but the pseudo-code for one basic implementation is shown in algorithm 1. The core of the algorithm lies in that it does not always move to a better solution, but rather it has a chance to accept worse states. The probability to do so depends on an ever-decreasing temperature, meaning that the

algorithm starts by spending more time exploring the function in the beginning when the temperature is high, and then increasingly focusing on exploitation as the temperature becomes lower. It is this feature that stops it from getting stuck in local minima, thus making it interesting for optimization with non-convex objective functions. A visualization of the algorithm in action in a very simple situation can be seen in figure 3.

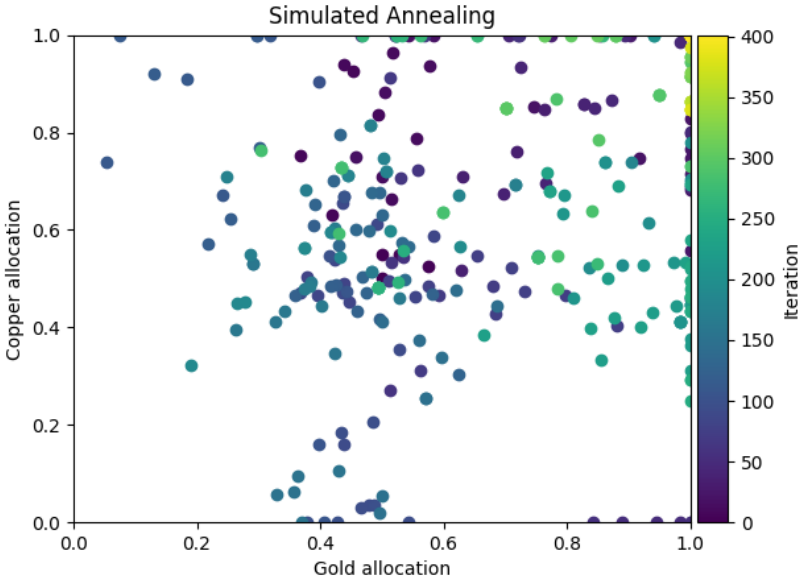


Figure 3: Visualization of simulated annealing when optimizing a simple portfolio consisting of two different futures contracts, both available at the same two markets

It can be noted that the algorithm itself is very straight-forward and easy to implement, with the base algorithm being short and concise. This corresponds well with the criterion that the algorithm should be simple to program, while at the same time also having a very low overhead.

As for the temperature scheme which decides how the new temperature will be calculated (called *cooling schedule*), abstracted as `newTemperature()` in algorithm 1, there exist a few alternatives [11]. In this study, the original exponential cooling schedule as proposed by Kirkpatrick, Gelatt, and Vecchi was used. The temperature during iteration n using this schedule can be seen in equation 3 and it depends on an additional constant, α, which decides the cooling speed.

$$T_n = \alpha^n T_0 \tag{3}$$

4.2 Differential evolution

Differential evolution (DE) is an evolutionary search algorithm that exist in many different variants and is well capable of optimizing non-convex and non-differentiable functions [12]. The algorithm can be described as a population-based, stochastic, and derivative-free.

Algorithm 2 Differential evolution (DE/rand/1)

$NP \geq 4$ ▷ Population size
$F \in [0, 2]$ ▷ Mutation factor
$CR \in [0, 1]$ ▷ Crossover probability
initialize all NP individuals $x \in \mathbb{R}^D$ randomly ▷ $\mathbb{R}^D$=Search space
$evaluations \leftarrow 0$
while $evaluations < maxEvaluations$ **do**
  for $i \leftarrow 1$ to NP **do**
    Choose three random distinct individuals x_{r1}, x_{r2}, x_{r3} s.t. $r_1 \neq r_2 \neq r_3 \neq i$

    $v_i \leftarrow x_{r1} + F(x_{r2} - x_{r3})$ ▷ Mutate
    for $j \leftarrow 1$ to D **do** ▷ Crossover
      $u_{i,j} \leftarrow v_{i,j}$ **if** (RANDOM$(0, 1) \leq CR$) **else** $x_{i,j}$
    end for
    if EVAL$(u_i) \leq$ EVAL(x_i) **then**
      $x_i \leftarrow u_i$
    end if
  end for
  $evaluations \leftarrow evaluations + NP$
end while
return best individual in population

Pseudo-code for the differential evolution variant denoted "DE/rand/1" can be seen in algorithm 2. Regardless of variant, all implementations of differential evolution function in the same general way. As it is population-based, it consists of a population of individuals that will be updated one generation at a time. Each individual resembles a vector in the search space, which in this case corresponds to one possible portfolio allocation of the financial contracts.

The idea is that all individuals in the population are initialized with random positions in the search space, or random portfolio allocations in this particular application, which will then balance exploring the search space with exploiting previous knowledge in order to find and move towards the best possible solution.

More specifically, during each iteration the algorithm will go through all individuals x_i and create a corresponding mutation vector v_i for each and all of them. Exactly how this mutation vector is formed is defined by the mutation scheme, but all schemes combines the current individual with a number of other individuals from the population, with at least some of them being randomly selected. A new candidate individual u_i is then formed by randomly taking some values from the current individual, and others from the formed mutation vector. If this candidate results in a better value, in this case a lower initial margin requirement, it will replace the current individual - else the candidate is simply discarded. Figure 4 shows this process in action.

Empirical results show that the performance of differential evolution can be heavily dependent on the configuration used, meaning that it is important both to choose appropriate values for the control parameters (NP, F and CR) and mutation scheme, depending on the nature of the function to be optimized [5].

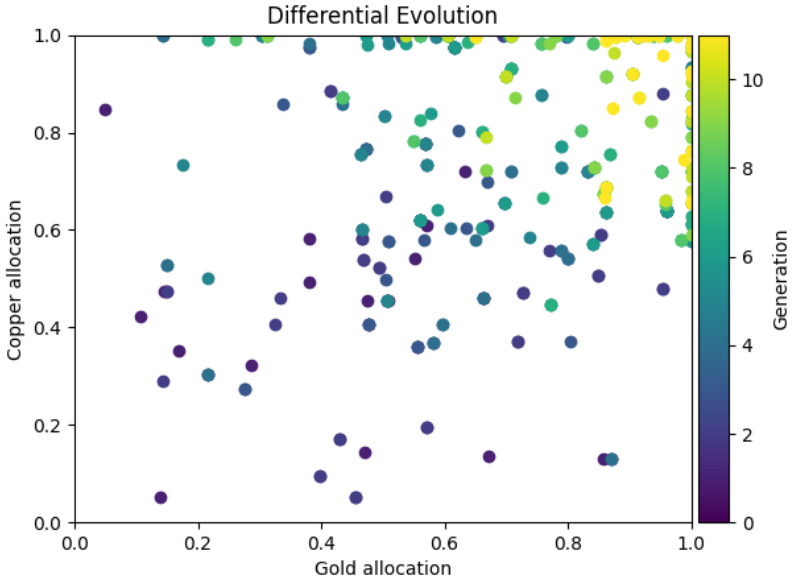


Figure 4: Visualization of differential evolution with population size 40 when optimizing a simple portfolio consisting of two different futures contracts, both available at the same two markets

4.2.1 Mutation schemes and parameters

There exist a number of mutation schemes, which determine how the mutant vector v_i is constructed (see algorithm 2). Some of the most common schemes can be seen in table 1.

Table 1: Some common differential evolution mutation schemes [12]

Name	**Mutation vector**
DE/rand/1	$v_i = x_{r1} + F(x_{r2} - x_{r3})$
DE/best/1	$v_i = x_{best} + F(x_{r1} - x_{r2})$
DE/rand/2	$v_i = x_{r1} + F(x_{r2} - x_{r3}) + F(x_{r4} - x_{r5})$
DE/best/2	$v_i = x_{best} + F(x_{r1} - x_{r2}) + F(x_{r3} - x_{r4})$
DE/current-to-best/1	$v_i = x_i + F(x_{best} - x_i) + F(x_{r1} - x_{r2})$
DE/current-to-rand/1	$v_i = x_i + rand(x_{r1} - x_i) + F(x_{r2} - x_{r3})$

As seen in the pseudo-code for the algorithm, there are a number of different parameters that need to be set, and there have been a number of research papers published investigating which values are appropriate.

First and foremost, NP is perhaps the most straightforward parameter, but finding the correct value for the problem at hand may still be far from trivial. Indeed, finding the optimal value for any given situation is a whole research topic in itself. Piotrowski has written a paper testing different values using various DE implementations in a number of situations, and using the results to outline a few guidelines for choosing an appropriate NP value [13]. In general, the number of individuals should increase as the dimensionality D of the problem or the allowed objective function evaluations increase. However, for

practical reasons a fixed population size was chosen instead, based on an estimation of the typical number of dimensions and evaluations available.

Secondly, the values chosen for the other parameters F and CR were based largely on the results from [14], with some preliminary testing conducted to verify that they provide at least decent performance in this case as well.

4.3 Particle swarm optimization

The particle swarm optimization (PSO) algorithm is a population-based optimization method with stochastic elements, having been developed by Kennedy and Eberhart in 1995 [15]. It is inspired by phenomena found in nature, more specifically the behaviour of bird flocks and fish schooling. The agents in its population, which are called *particles*, have a positions and velocity. Their movement is determined both by a deterministic part taking into account the best known location by the swarm, and a stochastic component to ensure sufficient exploration. The algorithm has since its development seen wide popularity and sparked interest in the research field of swarm intelligence [3, p. 21].

While PSO exist in many different variants, including hybrid methods combining it with other strategies [4, p. 1257], the pseudo code for one basic variant can be seen in algorithm 3. The core of the algorithm lies in that each particle has knowledge not only about its own best visited position, but also the best position seen by the swarm as a whole. During each update step the particles will update its velocity, moving towards both the particle's own best point, and the swarm's best point. How much it will move will depend on the distance to said points as well as a random factor.

This method will result in a behaviour where the particles start spread out in the search space, covering a large area in the beginning attempting to find the area of the global optimum, and then gradually converging to what is thought to be that area. As the time goes on, the swarm will converge towards that point and increasingly focus on a local search in the area, as seen in figure 5. This allows for effective coverage of non-convex functions, even when nothing is known about the function itself.

4.3.1 Population size and other parameters

The population size N is perhaps the most intuitive parameter used by the particle swarm optimization algorithm, but it can still be difficult to find good values. Just like with differential evolution, finding the optimal values is a whole research focus by itself. This study will not delve deeper into finding the best values for each situation, but instead base the parameters on some guidelines from existing research on the topic [16], [17].

The other three parameters used by PSO, W, C_p and C_g, are also quite straight-forward. They all decide the weights used when determining the new velocity of a particle. For example, C_g is a constant deciding how much the particles will be drawn towards the globally best known position. The values for these three parameters will be loosely based on the guidelines found in [17], but with some adjustments made to account for differences being made based on preliminary testing.

Algorithm 3 Particle swarm optimization

Choose values for W, C_p, C_g ▷ Constant coefficients
for $i \leftarrow 1$ to N **do** ▷ Initialize all particles
  initialize x_i to random position in search space ▷ Position
  $p_i \leftarrow x_i$ ▷ Particle's best position
  $g \leftarrow p_i$ **if** Eval$(p_i) <$ Eval(g) ▷ Swarm global best
  $v_i \leftarrow 0$ ▷ Particle velocity
end for

while *evaluations* $<$ *maxEvaluations* **do**
  for $i \leftarrow 1$ to N **do** ▷ N = Number of particles
    for $j \leftarrow 1$ to D **do** ▷ D = Number of dimensions
      $r_p \leftarrow$ random$(0, 1)$
      $r_g \leftarrow$ random$(0, 1)$
      $v_{i,j} \leftarrow W v_{i,j} + C_p r_p (p_{i,j} - x_{i,j}) + C_g r_g (g_j - x_{i,j})$
    end for
    $x_i \leftarrow x_i + v_i$
    if Eval$(x_i) <$ Eval(p_i) **then**
      $p_i \leftarrow x_i$ ▷ New particle best
      if Eval$(p_i) <$ Eval(g) **then**
        $g \leftarrow p_i$ ▷ New swarm best
      end if
    end if
  end for
  evaluations $\leftarrow$ *evaluations* $+ N$
end while
return g

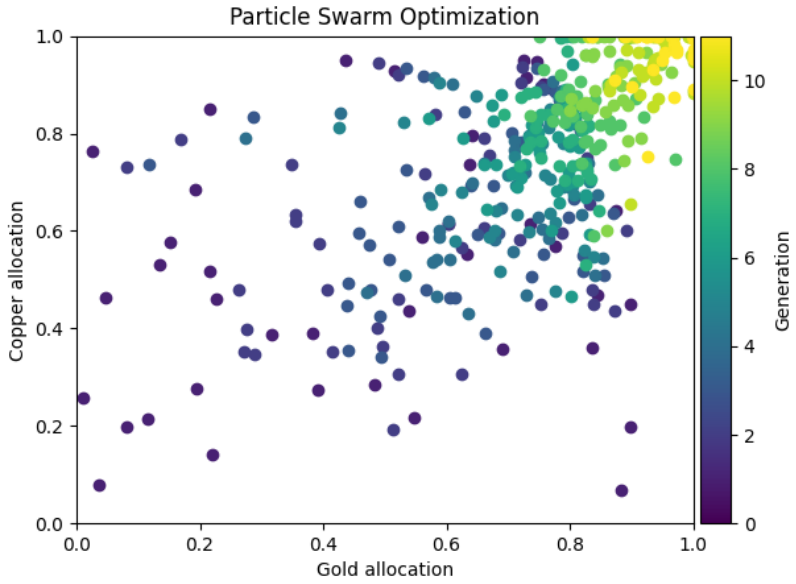


Figure 5: Visualization of particle swarm optimization with population size 40 when optimizing a simple portfolio consisting of two different futures contracts, both available at the same two markets

4.4 Other algorithms

There exist many more optimization algorithms than those that are evaluated in this report. Some algorithms have been deliberately omitted as they have been deemed unfit for the problem according to the evaluation criteria. Other algorithms have been left out simply because there exist too many optimization algorithms for it to be practical to find and evaluate them all, especially so given the vast number of recent nature-inspired meta-heuristic algorithms [18]. Therefore, an algorithm being left out from this paper does not automatically mean that it performs poorly or is unfit for this problem, on the contrary they may be just as good as or even better than the algorithms evaluated here.

5 Results

As the algorithms were implemented they were tested in a number of scenarios, and the results from those tests can be found in this section. Details about the different portfolios used for testing can be found in section 3.4.1.

5.1 Finding the solution in simple scenarios

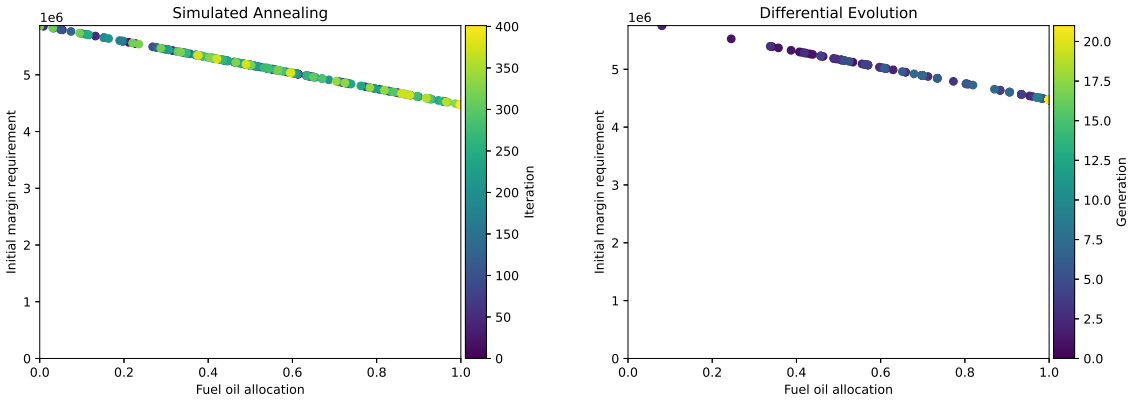
Both portfolio A and B constitute simple enough objective functions that their global optima can be found by a human, which makes it possible to test whether the optimizers are also able to find it as well. With the optimal allocation, initial margin requirements can be decreased by 23.74% and 62.41% for portfolio A and B, respectively.

Figure 6 shows the process of optimizing portfolio A using the different implemented optimizers. They were each given a budget of 400 objective function evaluations. It should be noted that the objective function takes on an extremely simple shape, essentially a straight line.

Portfolio B takes on a slightly more complex shape. It is also harder to visualize due to it being two-dimensional, but it can be noted from figure 7 that its shape roughly resembles a valley.

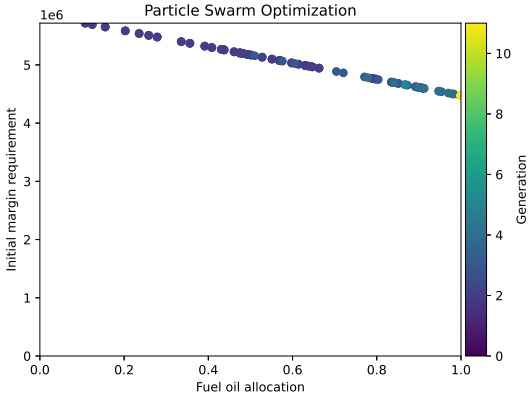
All optimizers were able to find the optimal solution (maximum deviation of 0.01% accepted) of portfolio A in 1,000 out of 1,000 test runs with as little as 120 objective function evaluations.

In the case of the slightly more complex portfolio B the optimizers are not always as reliable, as shown by table 2. Especially PSO has some difficulties finding the solution, although its performance does improve as it is given more evaluations. It should be noted that the optimizer may still get very close to the optimal solution even when it "fails", as the maximum deviation of 0.01% is a quite strict requirement.



(a) Simulated Annealing

(b) Differential Evolution



(c) Particle Swarm Optimization

Figure 6: Portfolio A optimization process with 400 objective function evaluations using different algorithms

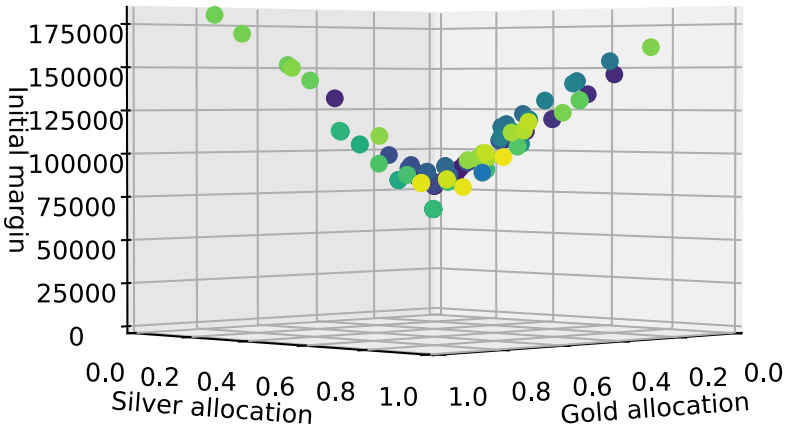


Figure 7: Process of optimizing portfolio B using simulated annealing

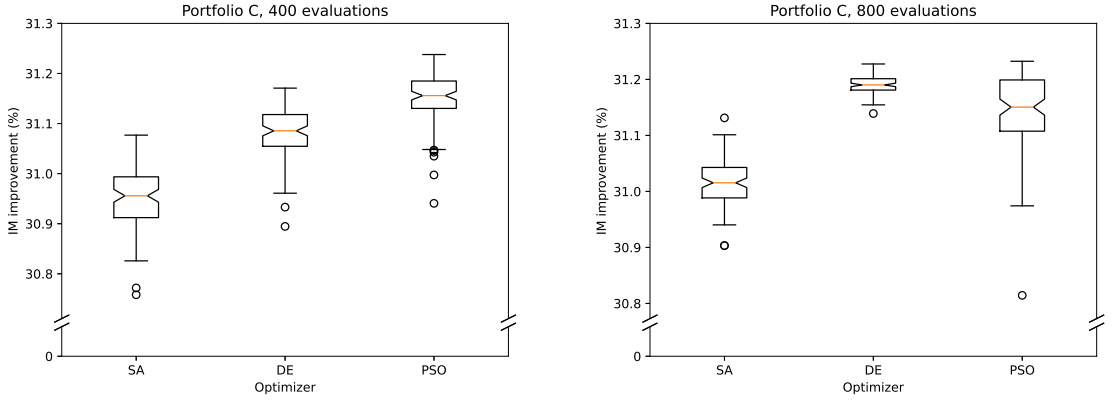
Table 2: Number of times (out of 1,000 attempts) the optimal allocation of portfolio B is found by each optimizer, given different numbers of maximum objective function evaluations

Evaluations	SA	DE	PSO
400	1,000	989	869
800	1,000	998	928
1,600	1,000	998	951

5.2 Scaling up the problem

Portfolios C, D and E all scale up the problem, increasing the number of dimensions. As the objective function becomes more complex with those portfolios, the optimal portfolio allocation is no longer known beforehand. Therefore the tests performed with those portfolios focus not whether on the solution is found, but rather on quantifying how much the IM requirement is minimized.

The results from optimizing portfolio C with 400 and 800 objective function evaluations, respectively, can be seen in figure 8. More complete results, including runs with 200 and 1,600 evaluations as well, can be found in appendix B.1.

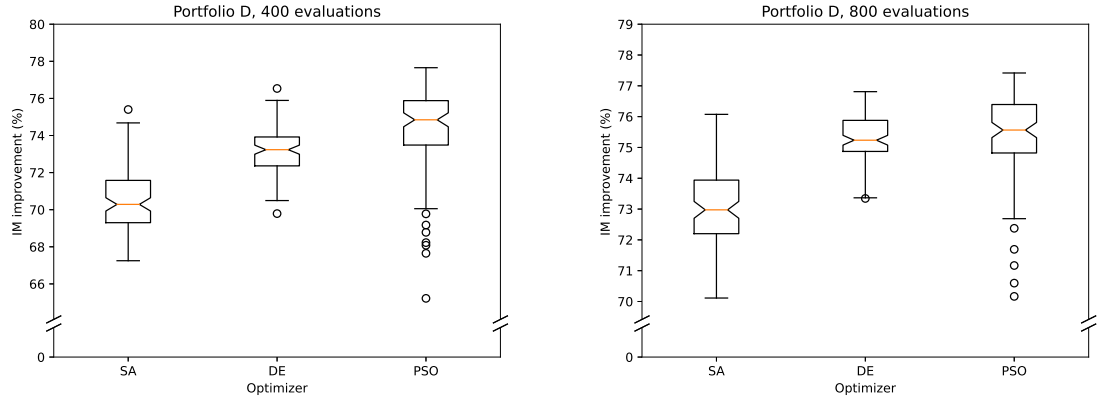


(a) 400 evaluations (b) 800 evaluations

Figure 8: Result from 100 optimization runs on portfolio C with different optimizers and maximum number of objective function evaluations

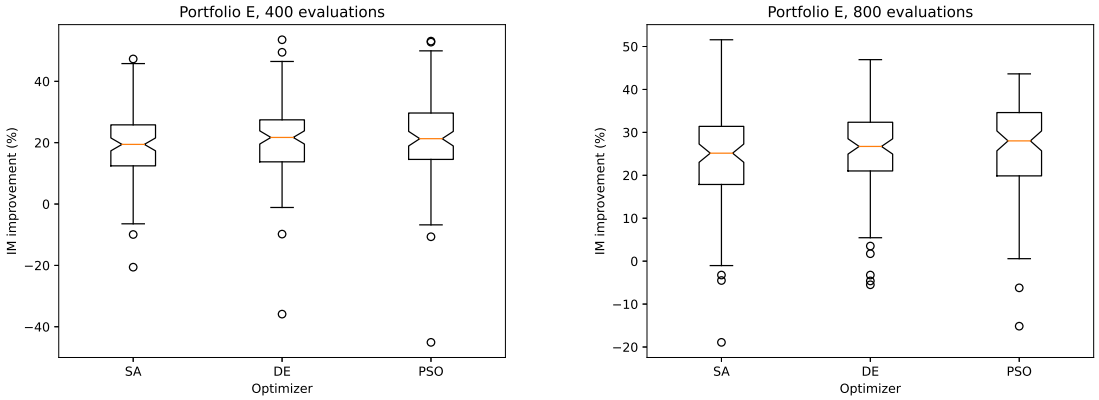
Similarly, the results from running the optimizers on portfolio D with 400 and 800 objective function evaluations can be found in figure 9. Additional data with 200 and 1,600 evaluations as well can be seen in appendix B.2.

The results from the random-generated portfolio E with 400 and 800 objective function evaluations are found in figure 10. As noted in section 3.4.1, the instrument positions are generated randomly before each test run, introducing a stochastic factor in the potential for optimization not present with the other portfolios. The full results, including 200 and 1,600 objective evaluations as well, can be seen in appendix B.3.



(a) 400 evaluations (b) 800 evaluations

Figure 9: Result from 100 optimization runs on portfolio D with different optimizers and maximum number of objective function evaluations



(a) 400 evaluations (b) 800 evaluations

Figure 10: Result from 100 optimization runs on portfolio D with different optimizers and maximum number of objective function evaluations

5.3 Execution time

Both qualitative and quantitative tests were conducted to provide insights into the temporal performance of the implemented algorithms. The qualitative test focuses on delving deeper into the optimization process and identifying which parts are the most time consuming. The quantitative tests attempt to provide information on how long the process as a whole takes in different situations, and whether there are differences between the algorithms.

Detailed timing information of the optimizers was generated based on a randomized scenario featuring 34 instruments spread among three markets, which can be seen in figure 11. The figure breaks down the total execution time of the optimization process into four distinct categories:

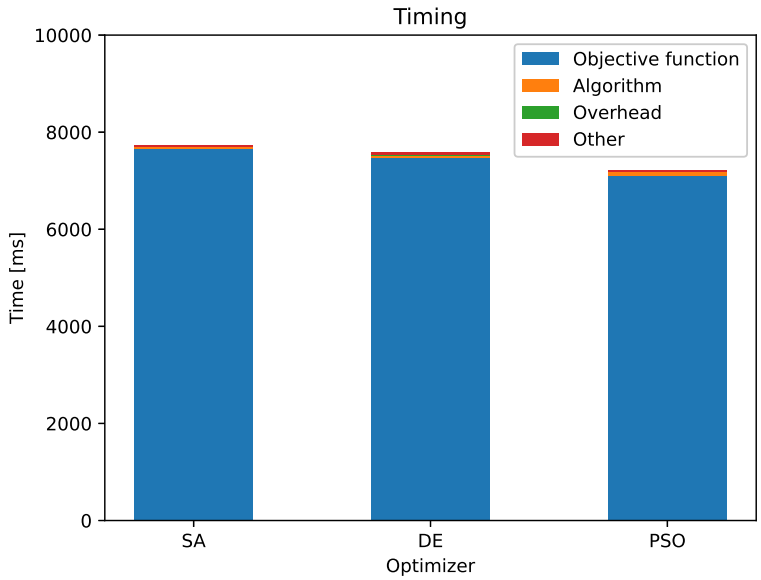


Figure 11: Timing of the different parts of the optimization process

1. Time spent evaluating the objective function (excluding time spent fetching from cache).
2. Time consumed by the optimization algorithms themselves.
3. Implementation overhead for tying different parts together.
4. Other time which could not be directly tracked down, but can probably be attributed to the algorithm and/or overhead.

As the timing numbers are based on very limited data they are quite unreliable, and as such the reader should be careful to make direct comparisons between the optimizers. What should be noted is however the difference in magnitude of the different parts of the optimization process. The overwhelming majority of the execution time is not consumed by any part of the optimizers themselves, but rather spent evaluating the objective function. Roughly 98% of the execution time is spent evaluating the objective function in this case.

The average execution time per evaluation when optimizing portfolios C, D, and E, can be seen in figures 12, 13, and 14, respectively. The figures show both the mean value (the circle) and the 95% confidence interval. Note that in no situation were any of the algorithms significantly faster or slower than the others. The figures also indicate that the evaluation time increases linearly as the number of evaluations increase, since the execution time per evaluation stays roughly the same as the number of evaluations increase. Furthermore, the execution time per evaluation increases as the dimensionality increases, although the rate of increase is not clear from the results.

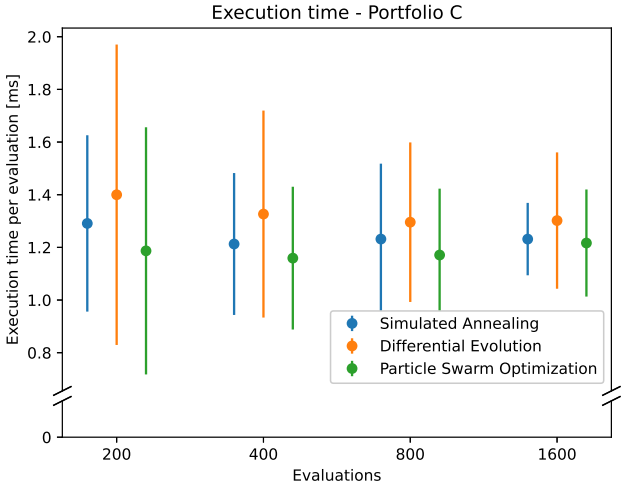


Figure 12: Execution time per evaluation when optimizing portfolio C

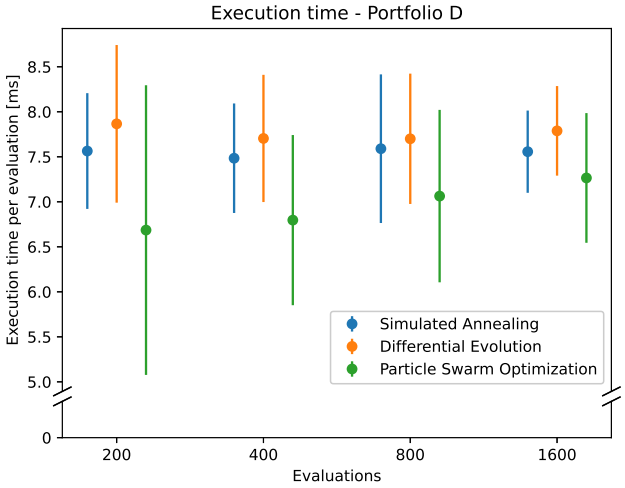


Figure 13: Execution time per evaluation when optimizing portfolio D

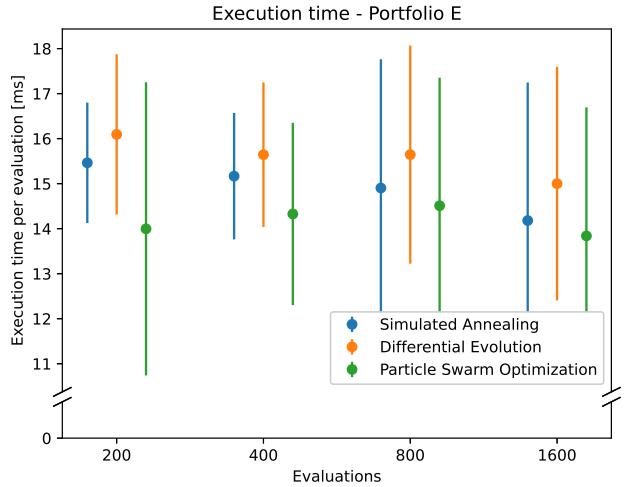


Figure 14: Execution time per evaluation when optimizing portfolio E

6 Discussion

This section discusses the limitations of the work, the potential for optimization in this situation, and advantages and disadvantages of the three algorithms evaluated. The idea is to provide some insights into the optimization process and provide some guidelines for the reader wanting to apply the knowledge gained from this project.

6.1 Limitations

Many parts of this project could have been investigated in far more detail, but limitations were necessary to accommodate for time constraints and keep the scope realistic. These limitations can broadly speaking be divided into one of two categories, algorithmic and financial.

In terms of algorithmic limitations it was necessary to minimize both the number of algorithms tested and the level of details with which they were investigated, due to time constraints. As there exist a vast number of optimization algorithms and often many variations of each and one of them, it is highly impractical to test them all. Thus only a select few were chosen and investigated, meaning that possibly superior algorithms may have been overlooked.

It was also not practical to search for the best possible variant of each algorithm, or the ideal parameters for each and one of them for that matter. Indeed, as shown by papers like [12]–[14], [16], [17], these things are whole research topics of their own. What this means however is that there may be additional potential with all of the investigated algorithms, in that they could potentially perform much better if more time was spent tweaking them.

As for the financial aspect, the primary limitations were mostly linked to limits with the test data in various ways. The fact that artificially constructed rather than real portfolios had to be used is important to keep in mind when analysing the results. It means that while this paper can provide good insights into the potential of using computational optimization for finding and exploiting IM minimization opportunities, the reader should be highly careful with making any conclusions about the magnitude of such opportunities. This becomes even more important when considering the fact that the instrument pairings may not be perfect, even as care was taken to choose instruments that were as similar as possible.

The nature of the test data also put some limits on which situations could be tested. It was not always possible to test all situations which could be challenging from a technical standpoint, such as multi-market portfolios with instruments that can be swapped between several markets. Combined with the fact that only SPAN markets with only futures contracts were tested, both being due to time constraints, this means that there may exist additional challenges that were not encountered during this project.

6.2 Potential for optimizing initial margin requirements

What the results indicate is that all three algorithms are well able to minimize the initial margin requirements by reallocating instruments, whenever there is a potential for doing so. However, what was also shown was that as the problem became increasingly complex, the optimizers needed more evaluations in order to reliably produce good results. Whereas with the simple portfolios A and B the global optima could reliably be found, in the more complex situations there is a much larger variance. There the optimizers rarely, if ever, seem to find the global optima. However, even if there is a larger variance in those situations the IM requirement still often gets lowered by a large amount. If finding the global optima is not the priority, the optimizers seem to still provide quite good results in medium-complex situations with relatively few evaluations available.

The only exception to this is the large, random portfolio E. In some cases the optimizers were not able to improve the objective value at all, only finding allocations that were worse than the initial one. It should, however, be noted that this portfolio was much larger than the other ones, and the results also did seem to improve as the number of evaluations increased. It should also be stated that the results are not directly comparable between the portfolios, since the potential for optimization differ.

Still, portfolio E signifies one potential obstacle. As the objective function becomes increasingly complex, the optimizers will require more evaluations to operate reliably. This may not be a large issue in itself, but the results also showed that the execution time per evaluation increased with the portfolio complexity. While the execution time for one optimization process was typically a couple of seconds for the medium-sized portfolios C and D, it could easily become a lot higher if the number of instruments and/or clearing houses increase. Therefore there may be a need for speeding up the process. For this there exist a number of options, including

- utilizing better hardware
- using better algorithms or improved variants of the algorithms investigated here
- optimizing the parameter values used
- parallelizing the algorithm implementations.

Another aspect to consider, while not directly related to the actual optimization process, is that creating the objective function is in this case far from trivial. In particular, finding good instrument pairings is far from trivial and was perhaps the most challenging part of the overall process. The semi-manual approach that was used in this project does not scale well as the number of instruments become much larger and complex. Therefore it is important to keep in mind that constructing the optimizer itself is only one part of the overall process, other parts such as creating the objective function should not be neglected.

To summarize, there seems to be a good potential for optimizing IM requirements as in most cases the optimizers were able to provide good results within only a couple of seconds at maximum. However, there also exist some possible obstacles, and there is an inevitable trade-off between accuracy and execution time.

6.3 Choosing which algorithm to use

According to the results there is not much difference between the algorithms in performance. They were all well able to decrease the IM, and no statistically significant difference in execution time could be found. The results do show slightly worse performance for simulated annealing than the other algorithms with portfolios C-E, though it is not clear whether this is due to the algorithm itself being worse or simply that it had slightly less optimal values for its control parameters. It should also be noted that SA was the most reliable at finding the global optima for portfolio B. However, if performance is so similar, it raises the question: Which algorithm should actually be chosen?

That question does not have a simple answer, but the quite similar performance in terms of relative IM improvement does show that it may be wise to instead look at other factors when deciding for an algorithm. Indeed, achieving only marginally lower IM requirement may not provide much benefit. Instead, other factors like ease-of-use, possible parallelization, and potential for improvement could be considered.

Simulated annealing has an advantage in that its control parameters are both fewer and less abstract, making the configuration process more forgiving for the user. On the other hand it is much harder to parallelize than either PSO or DE, due to being sequential in nature. With both PSO and DE it should be relatively easy to evaluate a complete generation in parallel, allowing for potentially huge performance improvements on multi-threaded systems.

The differences between PSO and DE, at least in their basic forms, are quite subtle. They provide similar strengths and weaknesses, with both being population-based metaheuristic algorithms. The results seem to show that DE is more consistent than PSO, although more research is needed to determine whether this is due to differences in algorithm or parameters. DE is also slightly simpler and easier to implement, although neither of them should pose any real challenge. DE does however have a disadvantage in that it may be slightly harder to parallelize, since updating one of its individuals requires the values of other individuals - introducing a race condition if not handled properly. Still, this is nothing that cannot be addressed, and both algorithms have good potential for parallelization.

Also, as stated previously there exist countless additional alternative algorithms, some of which can probably prove to be quite good for this application. Should this option be pursued, it can be good to keep in mind the fact that the vast majority of the execution time in this case was spent evaluating the objective function. As such, it is very likely that it is much more beneficial to reduce the number of objective function evaluations needed to achieve good results, rather than reducing the overhead of the implementations.

In summary it comes down to first identifying the needs of the situation. Then, an algorithm can be chosen whose strengths and weaknesses fit together well with those needs. There is no silver bullet solution, no single algorithm which will always be the best choice.

7 Conclusion and future work

In this project, the possibility for using computational optimization in order to minimize initial margin requirements was investigated. In particular, the project focused on whether the algorithms tested were able to find and exploit potential for optimization, whether they are able to do so in a reasonable time frame, and what the differences between the different implementations were. It was found that all three implemented algorithms - simulated annealing, differential evolution, and particle swarm optimization - performed well in the scenarios tested. As such, the results indicate that there is a good potential for optimization and that it seems to be practical to apply any of these algorithms in a real world scenario.

There are, however, also some caveats with the results that should be kept in mind and preferably looked into in more detail. These are mostly related to limits in the test data which creates some uncertainty in the result, but it was also shown that in more complex situations there may be a need to speed up the optimization process in order to achieve good results in reasonable time. Given those caveats, there are many ways in which this project could be expanded upon to provide more insights into the subject.

It could be interesting to test more algorithms or more variants of the existing algorithms, as well as additional parameter choices. It would also be interesting to run the tests in more situations, perhaps with additional types of instruments, multi-way instrument pairings, and with clearing houses using more types of methods for calculating IM requirements. That would provide a more complete picture of the situation, and as such probably provide slightly more definitive answers to which approaches are good.

If the optimizers are to be applied in a real situation, depending on the situation it may be necessary to speed up the process. Except testing the previously mentioned things to perhaps find slightly "smarter" optimizers, making increased use of parallelization is probably one of the main areas of potential improvement. Therefore looking into how the different algorithms can be parallelized, which benefits and challenges lie therein, and quantifying just how much the process can be sped up with parallelization is one possible topic for future work.

Another thing that can be investigated is if it is possible to make the optimizer better at adhering to the wishes of the investor. For example, an investor probably wants to limit the number of trades to minimize transaction costs. This could be done by expanding the functionality of the penalty term, penalizing changes in the portfolio. Perhaps the investor also wants to prioritize some markets over others, only deviating from the "favourites" if the IM reduction potential is large enough. The optimal way of doing this is likely non-trivial, and warrants further investigation.

A related topic which could be highly beneficial to investigate further is how instruments are paired with each other. This is not part of the optimization process itself, but it is still a highly connected topic since having pairings is necessary for actually having something to optimize. Being able to find the pairings in a fully automatic, data-driven way is more or less necessary in order to apply the optimizers in a broader setting, lest many man-hours are to be spent manually finding pairings.

8 References

[1] J. C. Hull, *Options, Futures, and Other Derivatives*, 11th ed. Pearson Education Limited, 2022, ISBN: 978-1-292-41065-4.

[2] CME Group. "CME SPAN, Standard Portfolio Analysis of Risk." (2010), [Online]. Available: https://www.cmegroup.com/clearing/files/span-methodology.pdf (visited on 02/22/2023).

[3] X.-S. Yang and S. Koziel, *Computational Optimization, Methods and Algorithms*, 1st ed. 2011, vol. 356, ISBN: 978-3-642-20858-4. DOI: 10.1007/978-3-642-20859-1.

[4] L. Rios and N. Sahinidis, "Derivative-free optimization: A review of algorithms and comparison of software implementations," *Journal of Global Optimization*, vol. 56, Nov. 2009. DOI: 10.1007/s10898-012-9951-y.

[5] A. Auger, N. Hansen, J. Pérez Zerpa, R. Ros, and M. Schoenauer, "Experimental comparisons of derivative free optimization algorithms," Jun. 2009, pp. 3–15, ISBN: 978-3-642-02010-0. DOI: 10.1007/978-3-642-02011-7_3.

[6] J. Larson, M. Menickelly, and S. M. Wild, "Derivative-free optimization methods," Apr. 2019. DOI: 10.1017/S0962492919000060.

[7] D. Wolpert and W. Macready, "No free lunch theorems for optimization," *IEEE Transactions on Evolutionary Computation*, vol. 1, no. 1, pp. 67–82, Apr. 1997, ISSN: 1941-0026. DOI: 10.1109/4235.585893.

[8] R. Gamperle, S. Muller, and A. Koumoutsakos, "A parameter study for differential evolution," *Advances in Intelligent Systems, Fuzzy Systems, Evolutionary Computation*, vol. 10, pp. 293–298, Aug. 2002.

[9] M. Centeno-Telleria, E. Zulueta, U. Fernandez-Gamiz, D. Teso-Fz-Betoño, and A. Teso-Fz-Betoño, "Differential evolution optimal parameters tuning with artificial neural network," *Mathematics*, vol. 9, no. 4, 2021, ISSN: 2227-7390. DOI: 10.3390/math9040427. [Online]. Available: https://www.mdpi.com/2227-7390/9/4/427.

[10] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi, "Optimization by simulated annealing," *Science*, vol. 220, no. 4598, pp. 671–680, 1983. DOI: 10.1126/science.220.4598.671. eprint: https://www.science.org/doi/pdf/10.1126/science.220.4598.671. [Online]. Available: https://www.science.org/doi/abs/10.1126/science.220.4598.671.

[11] T.-C. Publishing. "A comparison of cooling schedules for simulated annealing." (), [Online]. Available: http://what-when-how.com/artificial-intelligence/a-comparison-of-cooling-schedules-for-simulated-annealing-artificial-intelligence/ (visited on 04/25/2023).

[12] M. Georgioudakis and V. Plevris, "A comparative study of differential evolution variants in constrained structural optimization," *Frontiers in Built Environment*, vol. 6, 2020, ISSN: 2297-3362. DOI: 10.3389/fbuil.2020.00102. [Online]. Available: https://www.frontiersin.org/articles/10.3389/fbuil.2020.00102.

[13] A. P. Piotrowski, "Review of differential evolution population size," *Swarm and Evolutionary Computation*, vol. 32, pp. 1–24, 2017, ISSN: 2210-6502. DOI: https://doi.org/10.1016/j.swevo.2016.05.003. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S2210650216300268.

[14] M. E. H. Pedersen, "Good parameters for differential evolution," *Magnus Erik Hvass Pedersen*, vol. 49, 2010.

[15] J. Kennedy and R. Eberhart, "Particle swarm optimization," in *Proceedings of ICNN'95 - International Conference on Neural Networks*, vol. 4, 1995, 1942–1948 vol.4. DOI: 10.1109/ICNN.1995.488968.

[16] A. P. Piotrowski, J. J. Napiorkowski, and A. E. Piotrowska, "Population size in particle swarm optimization," *Swarm and Evolutionary Computation*, vol. 58, p. 100 718, 2020, ISSN: 2210-6502. DOI: https://doi.org/10.1016/j.swevo.2020.100718. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S2210650220303710.

[17] M. E. H. Pedersen, "Good parameters for particle swarm optimization," *Hvass Lab., Copenhagen, Denmark, Tech. Rep. HL1001*, pp. 1551–3203, 2010.

[18] I. F. Jr., X. Yang, I. Fister, J. Brest, and D. Fister, "A brief review of nature-inspired algorithms for optimization," *CoRR*, vol. abs/1307.4186, 2013. arXiv: 1307.4186. [Online]. Available: http://arxiv.org/abs/1307.4186.

A Algorithm details and parameter values

The parameters used and other details of the implemented algorithms are mentioned here. These values were used for all tests unless otherwise stated. In general, the values were chosen primarily based on existing research (see section 4), and secondarily based on limited preliminary testing. As the test cases had between 1 and 222 dimensions and were given roughly 1,000 objective function evaluations, the values were chosen with this in mind. It should also be noted that the objective function allows inputs in the $[0, 1]$ range, as that also affects the values chosen.

A.1 Simulated annealing

The initial temperature used by the simulated annealing implementation was set depending on the original initial margin requirement, i.e. the objective function value prior to optimization. The aim was to set the initial temperature approximately equal to the maximum possible difference in value between two points in the search space, but this can vary greatly depending on the case. As such, it was approximated very roughly that the maximum possible difference is equal to half the original IM.

There is also the matter of how neighbours were chosen. According to the preliminary tests, using a Gaussian distribution with a standard deviation of 0.4 from the current value seemed to perform decently. Other alternatives that were investigated but which performed more poorly were a) uniform distribution within a certain range from the current value, and b) global uniform distribution.

- Initial temperature $(T_0) = IM_{pre-optimization}/2$
- Cooling factor, alpha $(\alpha) = 0.995$
- Neighbour standard deviation $(\sigma) = 0.4$

A.2 Differential evolution

For the differential evolution optimizer implementation, the mutation scheme "DE/rand/1" was used. As for the crossover part of the algorithm, a fine-grained approach was used where the crossover was decided for each individual instrument-portfolio pair (as opposed to applying the crossover on an instrument as a whole).

The following parameter values were used:

- Population size $(NP) = 20$
- Mutation factor $(F) = 0.9$
- Crossover probability $(CR) = 0.7$

A.3 Particle swarm optimization

The following parameter values were used for the PSO implementation:

- Population size (N) = 40
- Velocity modifier (W) = -0.4
- Particle best attraction modifier (C_p) = -0.6
- Global best attraction modifier (C_g) = 2.0

B Full results

B.1 Portfolio C

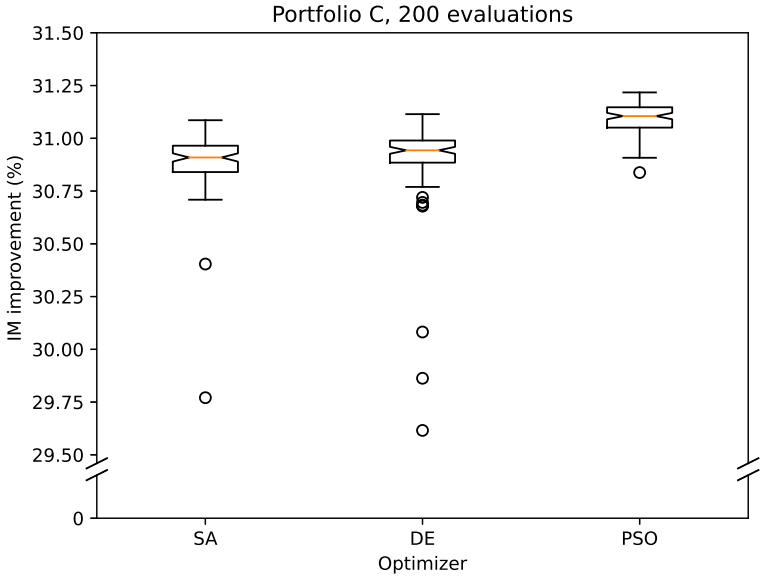


Figure 15: Result from 100 portfolio C optimization runs, each given 200 objective function evaluations per optimizer

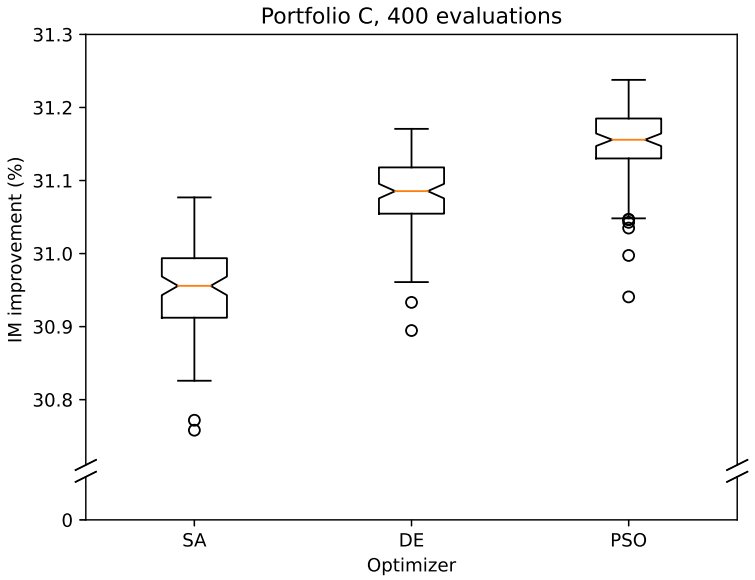


Figure 16: Result from 100 portfolio C optimization runs, each given 400 objective function evaluations per optimizer

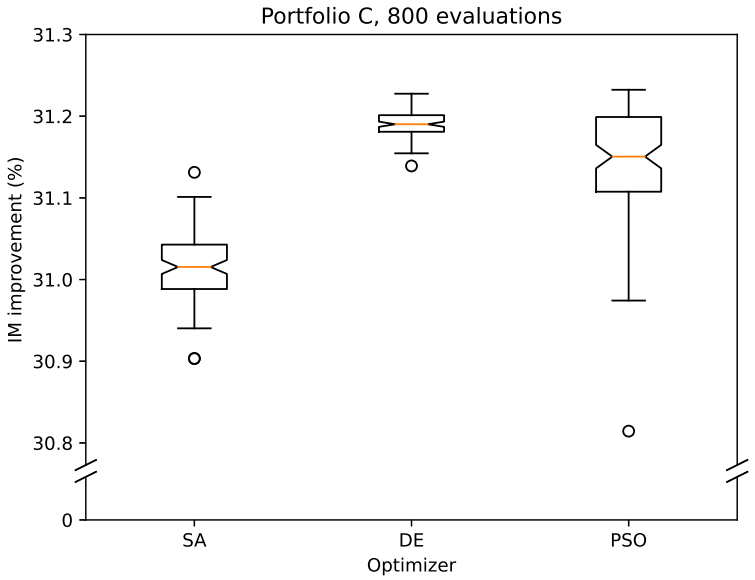


Figure 17: Result from 100 portfolio C optimization runs, each given 800 objective function evaluations per optimizer

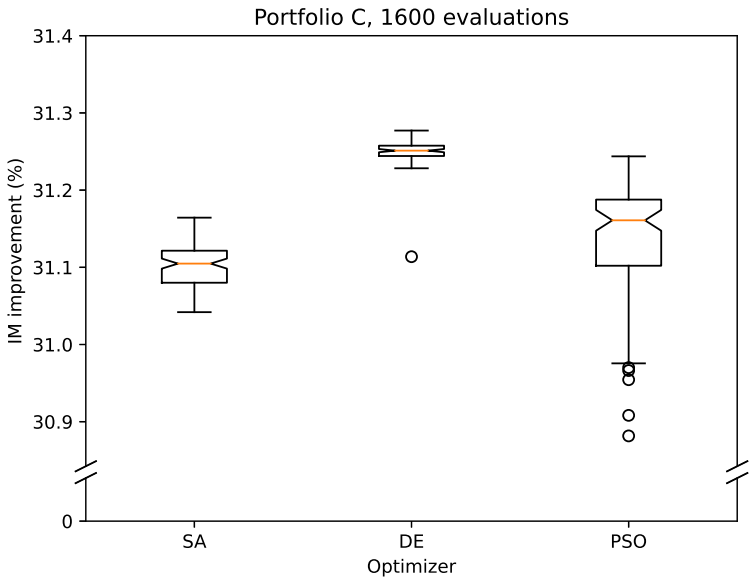


Figure 18: Result from 100 portfolio C optimization runs, each given 1,600 objective function evaluations per optimizer

B.2 Portfolio D

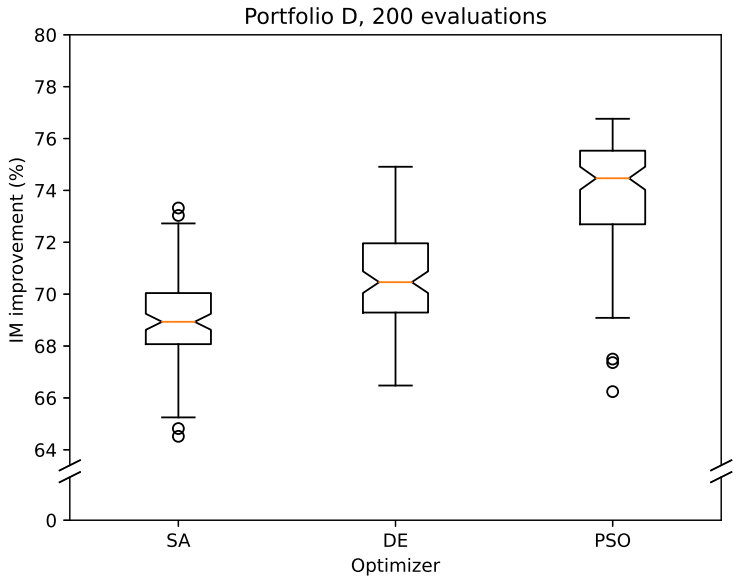


Figure 19: Result from 100 portfolio D optimization runs, each given 200 objective function evaluations per optimizer

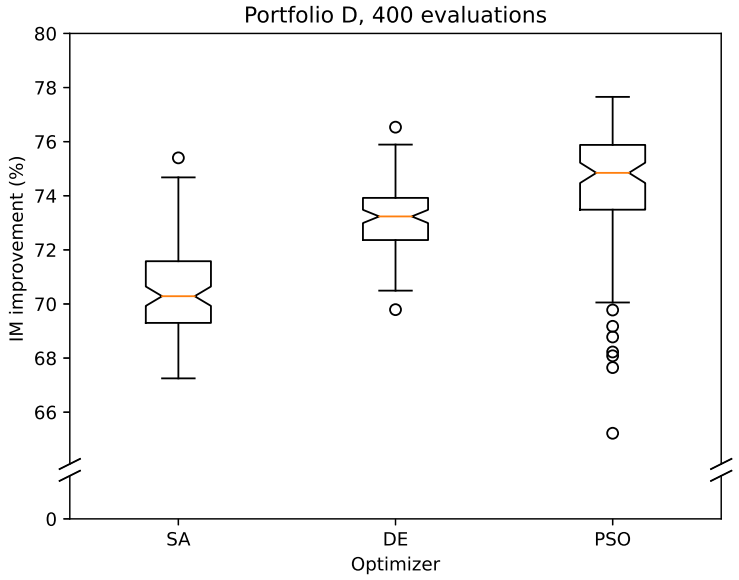


Figure 20: Result from 100 portfolio D optimization runs, each given 400 objective function evaluations per optimizer

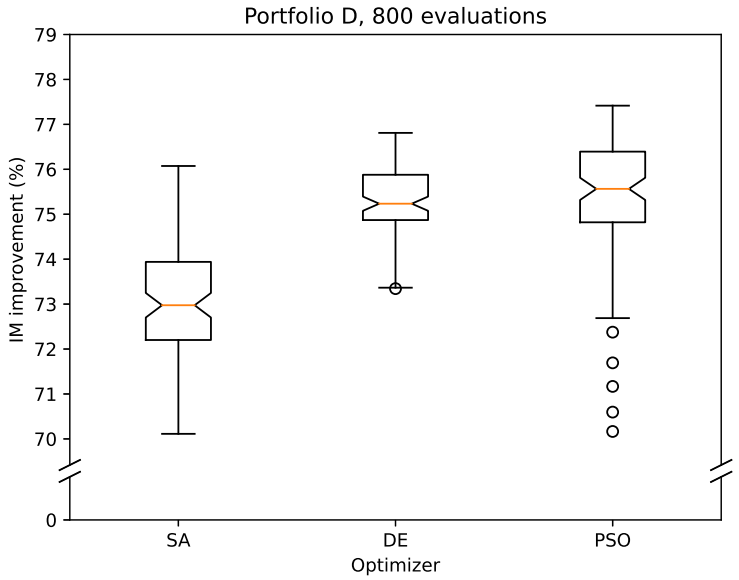


Figure 21: Result from 100 portfolio D optimization runs, each given 800 objective function evaluations per optimizer

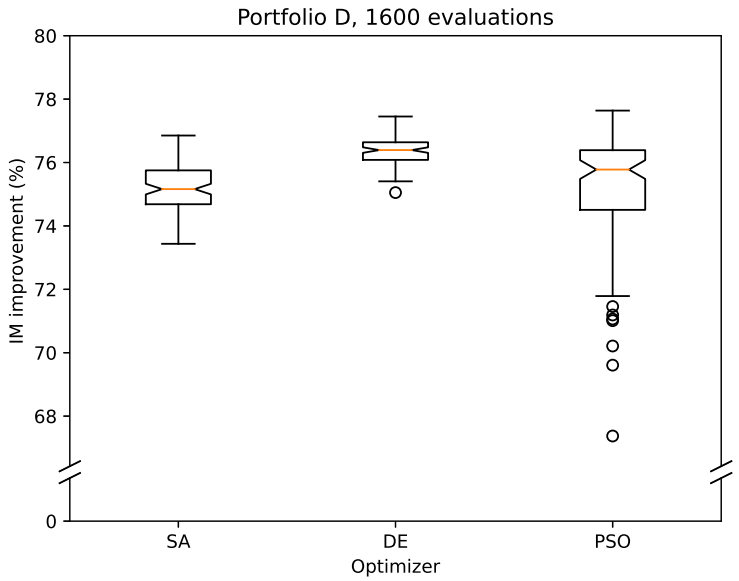


Figure 22: Result from 100 portfolio D optimization runs, each given 1,600 objective function evaluations per optimizer

B.3 Portfolio E

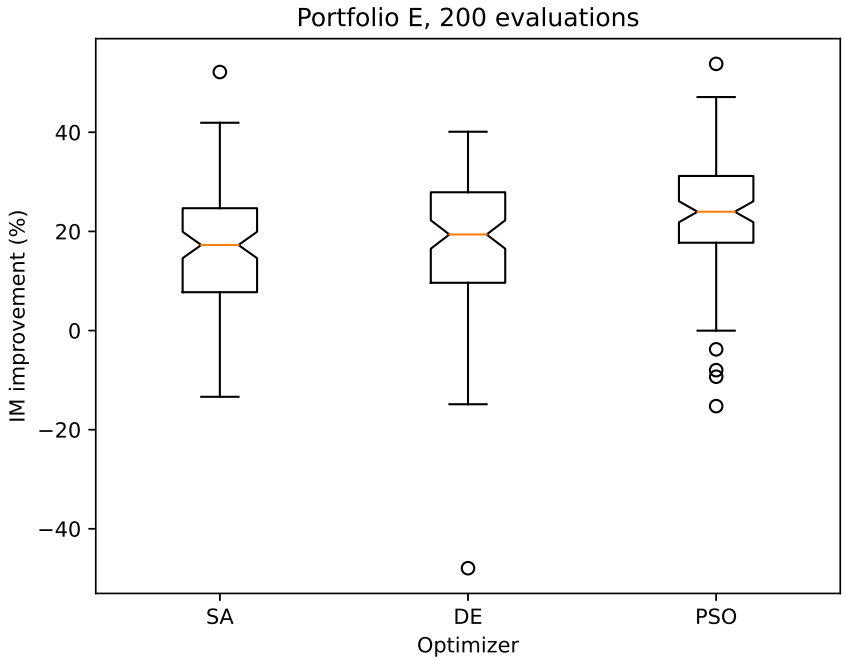


Figure 23: Result from 100 portfolio E optimization runs, each given 200 objective function evaluations per optimizer

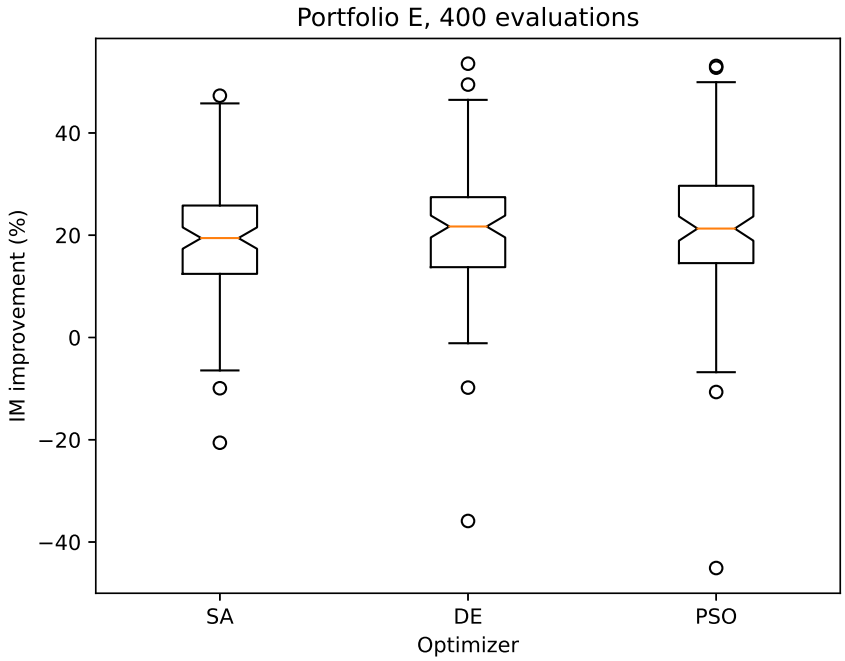


Figure 24: Result from 100 portfolio E optimization runs, each given 400 objective function evaluations per optimizer

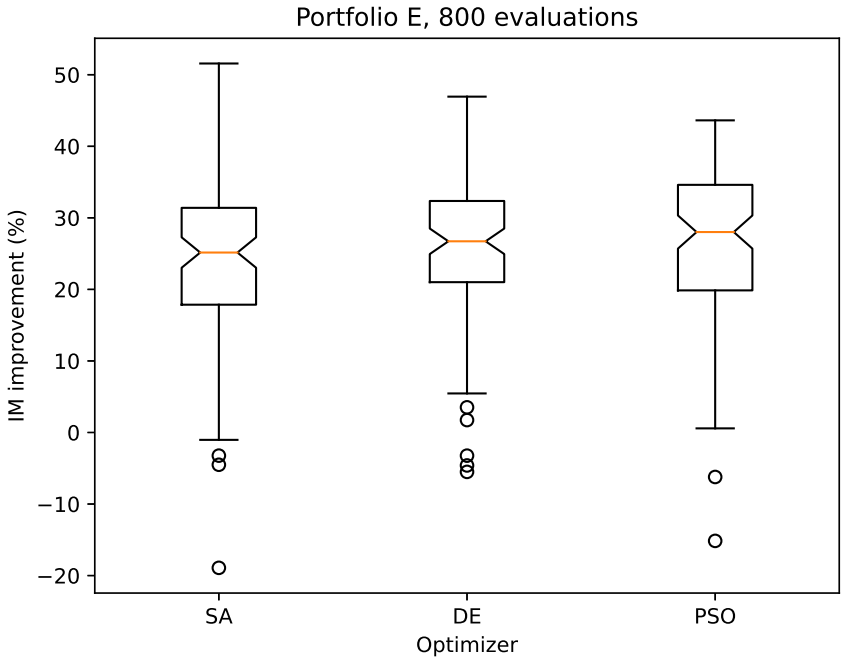


Figure 25: Result from 100 portfolio E optimization runs, each given 800 objective function evaluations per optimizer

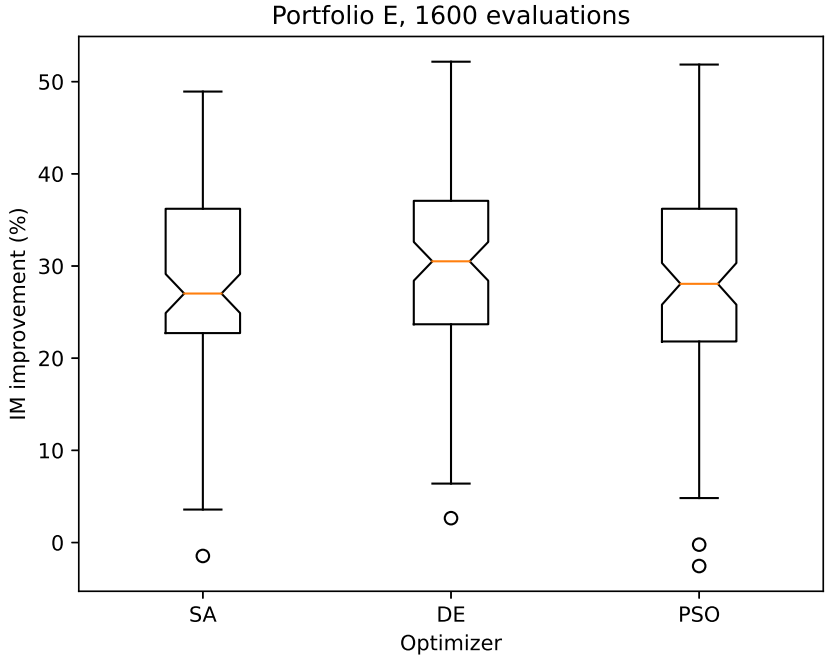


Figure 26: Result from 100 portfolio E optimization runs, each given 1,600 objective function evaluations per optimizer