University of Aberdeen

Department of Computing Sciences

CS4526 Project Report

2013–2014

Particle Swarm Optimisation for the Portfolio Selection Problem in a Function Based Environment.

Author : **Anthony S. Chapman**

Supervisor : **Dr. Wei Pang**

Declaration

I hereby declare that this report has been composed by me. I also declare that all sources of information have been specifically acknowledged and all quotations distinguished by quotation marks.

(signed) ....

Abstract

Abstract....

Acknowledgments

Acknowledgments....

Contents

CHAPTER 1

Introduction

CHAPTER 2

Background

In order to expand and adapt existing Particles Swarm Optimisation methods an initial background research has to be done to fully understand the concepts involved. Similarly to fully

1. Particle Swarm Optimisation

Particle swarm optimization (PSO) is a population based stochastic optimization technique developed by Kennedy and Eberhart in 1995, discovered through simplified social model simulation [**5, 13, 3, 10**]. It simulates the behavior of bird flocking involving the scenario of a collection of birds randomly looking for food in a search space. None of the birds know where the food is located, but they do know how far the food location is from their current positions. An effective technique for the bird to find food is to adjust their velocity according to the bird which is nearest to the food. PSO was motivated from this scenario and was developed to solve complex optimization problems, where the optimum position of the fitness function is where the food is located and all the birds are the particles searching for this optimum position.

In the conventional PSO, the behaviour displays particles in a multidimensional space where each particles has two properties: a position vector and a velocity vector. Let n be the number of particles in the swarm. Then each particles $i \in n$ has the properties as shown in (2.1):

$$\begin{aligned}
&V_i^k : \text{The velocity of particle i at time k.}\\
&X_i^k : \text{The current position of particle i at time k.}\\
&Pbest_i^k : \text{The personal best position of particle i at time k.}\\
&Gbest^k : \text{The global best position of particle i at time k.}
\end{aligned} \tag{2.1}$$

At each step, the velocity of the *ith* particle will be updated according to the following equation:

$$V_i^{k+1} = \omega V_i^k + c_1 r_1 \times \left(Pbest_i^k - X_i^k\right) + c_2 r_2 \times \left(Gbest^k - X_i^k\right)$$

where

$$\begin{aligned}
&V_i^{k+1} : \text{The velocity of particle i at time k + 1.} \\
&V_i^k : \text{The velocity of particle i at time k.} \\
&X_i^k : \text{The current position of particle i at time k.} \\
&\omega : \text{Inertia weight parameter.} \\
&c_1, c_2 : \text{Acceleration coefficients.} \\
&r_1, r_2 : \text{Random numbers between 0 and 1.} \\
&Pbest_i^k : \text{The personal best position of particle i at time k.} \\
&Gbest^k : \text{The global best position of particle i at time k.}
\end{aligned} \tag{2.2}$$

In the updating process 2.2 the acceleration coefficients c_1, c_2 and the inertia weight ω are predefined, and r_1, r_2 are uniformly generated random numbers in the range [0,1].

```
Initialisation
for i = 1,...,S do
    initPosition(i)
    initBestLocal(i)
    if i = 1 then
        initBestGlobal()
    end
    if improvedGlobal(i) then
        updateGlobalBest(i)
    end
    initVelocity(i)
end
Main program
while not endingCondition() do
    for i - 1,..., S do
        createRnd(r1, r2)
        updateVelocity(i, r1, r2)
        updatePosition(i)
        if improvedLocal(i) then
            updateBestLocal(i)
        end
        if improvedGlobal(i) then
            updateGlobalBest(i)
        end
    end
end
```

Algorithm 1: PSO pseudo-code.

In Algorithm 1 one can see the pseudo-code for a standard PSO. A step by step explanation of Algorithm 1 is as follows: First, there is an initialisation for all the particles in the PSO, for every particle i, initPosition(i) randomly created a particle with a designated position in the search space. For the first step, initBestLocal(i) = initPosition(i), but it will be updated afterwards. Then the *if* function makes a first global best and if any other particles has a better global best position it will be set as the new global best. initVelocity(i) gives particles i an initial velocity which is randomly generated.

After the initialisation, the core of the PSO method is executed until the ending-Condition() is satisfied, which can be the number of iterations or an improvement threshold. In the body of the *While* loop all particles are updated. The first step is to generate random numbers used in the velocity Equation (2.2). Then the actual velocity is updated. In the next step, updatePosition(i) updates the position of a particles in the search space and checks the fitness function value for improvement or not. A the end of the *for* loop, if improvedLocal(i) = True then updates the local best position and finally it is similar for updating the global best position.

2. Portfolio Management

The money maker...

3. Haskell

It rules!!

CHAPTER 3

Related Work

1. Markowitz Model

One of the first contributions to the portfolio problem was made by Markowitz [**8**] which was later described in more detail in his book [**9**]. Markowitz introduced the mean-variance model which considers the variance of the portfolio as the measure of the investor's risk under a set of assumptions. According to Markowitz, the portfolio selection problem can be expressed as an objective function subject to linear constraints.

Following Markowitz, the investment horizon includes a single period whereas the investment strategy is to select an optimal portfolio at the beginning of the period which will be held unchanged until the date of termination. The joint distribution of one-period security returns is assumed to be a multivariate normal distribution, which in turn follows that the distribution of the portfolio return is also normal.

Let $S = (S_1, S_2, S_3, \ldots, S_N)$ be the set of N assets where each asset S_i has a rate of return represented by a variable R_i with an expected return r_i and each pair of assets (S_i, S_j) has a covariance σ_{ij} . The variance-covariance matrix $\sigma_{n\times n}$ contains all the covariance values, furthermore, it is a symmetric matrix and every diagonal element σ_{ii} represents the variance of asset S_i. Let π be a positive value which represents the investor's required expected return. Generally the values r_i σ_{ij} are estimated from past data and are fixed during the period of investment.

According to Markowitz, a portfolio is a real valued vector $X = (x_1, x_2, x_3, \ldots, x_i)$ where each variable x_i represents the percentage invested in a corresponding asset S_i. The value $\sum\limits_{i=1}^{N}\sum\limits_{j=1}^{N} x_i x_j \sigma_{ij}$ is the variance of the portfolio and it is the measure of risk associated with the portfolio. From this, we may obtain a constrained portfolio optimisation problem:

$$\begin{aligned} &Minimise \\ &\sum_{i=1}^{N}\sum_{j=1}^{N} x_i x_j \sigma_{ij} \\ &Constraints: \\ &\sum_{i=1}^{N} x_i r_i = \pi \\ &\sum_{i=1}^{N} x_i = 1 \\ &x_i \geq 1, i \in \{1, 2, \ldots, N\} \end{aligned} \tag{3.1}$$

Here, the objective function minimises the total variance (risk) associated with a portfolio whilst the constraints ensure that the portfolio meets the expected required return of π and the proportions of all the assets sum to 1. The non-negative constraint means that no short-selling is allowed.

This framework presumes that the investor uses a quadratic utility function or that asset returns follow a multivariate normal distribution. This implies that the rate of return of a portfolio of assets can be completely explained by the expected return or variance of assets. The efficient-frontier is the set of assets which provide minimum risk for a specified level of return. Solving the Markowitz portfolio optimisation problems for different specified expected portfolio returns will give us a set which represents the efficient-frontier, it is a smooth semi-increasing curve which represents the best possible trade-off between risk and expected return, also called the set of Pareto-optimal portfolios [].

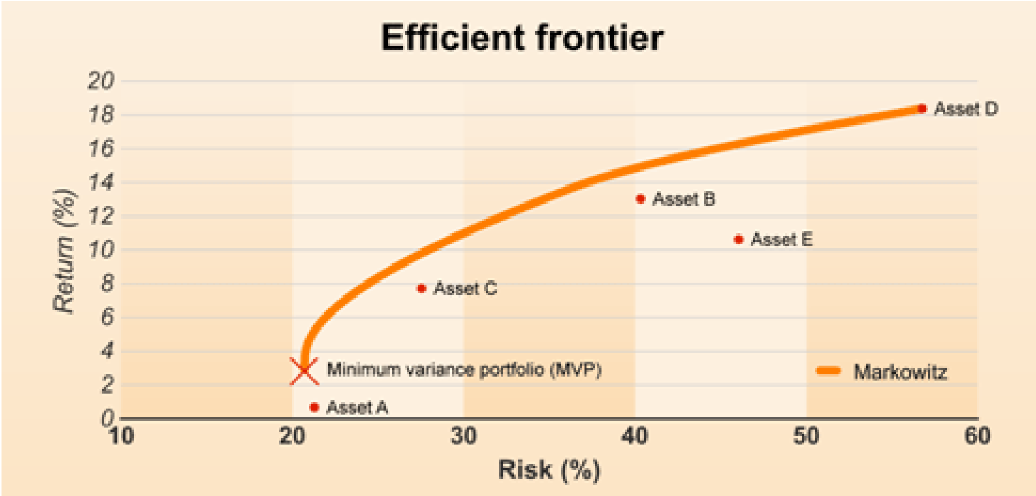

Figure 1. Example of an Efficient Frontier Without a Risk-Free Asset.

Each point on the line in Figure 1 represents a portfolio which is considered to be efficient (ie no other portfolio provides higher return without more risk and similarly for risk). In Figure 1 Assets A-E represent what the portfolio will be made out of.

Markowitz's model is subject to serous criticisms as stated in [**15**], and the main ones being that a measure of dispersion can be adopted as a measure of risk only if the relevant distribution is symmetric. Another problem is that the distribution of individual asset returns has a tendency to show a higher probability of being fat-tailed. In case of non-normal, non-symmetric distributions, the utility function must be quadratic [**15**].

This criteria should not be taken lightly since the assets' return does not follow normal distributions in real world situations [**7**]. This approach can only be used if the investor's utility function is quadratic with non-positive second derivatives or if the asset's return distribution can be fully described. Several research have indicated that that the quadratic utility function implies that beyond some level or return, marginal utility of the decision maker for wealth becomes negative [**1, 4**].

2. Portfolio in Excel

3. Something PSO

4. PSO applied

CHAPTER 4

Problem Domain

1. Approach

CHAPTER 5

Requirements

This section describes the requirements for this project. Table 1 refers to the functional requirements from technical point of view. Section 2 focuses on the non-functional requirements of the system.

1. Functional

No.	Description	Priority
1	**Optimisation of Portfolio**	
1.1	**PSO**	
1.1.1	Initialisation of particle population	High
1.1.2	Processing swarm optimisation	High
1.1.3	Updating the local and global (at each step) particle values	
1.1.4	Calculating an optimal solution	High
1.1.5	Presenting the results	High
1.2	**PSO for portfolio problem**	
1.2.1	Minimise portfolio variance	High
1.2.2	Maximise portfolio expected return	Low
1.2.3	Use multi-objective for optimum solution	Low
1.2.4	Refining results output	High
1.2.5	Make results for readable for user	High
2	**User Input**	
2.1	Allow the user to enter the name of the data file	High
2.2	Allow the user to change the expected portfolio return	High
2.3	Allow the user to select the name for the output file	Low
2.4	Allow the user to change the PSO particle size	Low
2.5	Allow the user to change the PSO iteration number	Low
3	**Output format**	
3.1	Display the results during run-time	High
3.2	Make results more readable for output file	High
3.3	Store results into a separate file	High

TABLE 1. Functional requirements for system.

2. Non-functional

As this system is an extension on a PSO module [**12**], it is crucially important to devote a considerable amount of time to testing. This is to ensure that the alterations do not affect the performance of the overall efficiency of the algorithm and quality of the optimisation.

The system's scalability is something not to be overlooked. As each asset in a portfolio represents one dimension in the fitness function (not to be confused with just another linear factor of the same coefficient in a function), optimising a function in, for example, 100 dimensions (100 assets) might be to much for the system to cope with.

Running the PSO requires setting up various parameters and thresholds for optimisation (size of the particle population, number of iterations, inertia weights and convergence coefficients). These parameters need to be optimised for the algorithm to be computationally effective and produce accurate results.

CHAPTER 6

Methodology and Technologies

This chapter describes the methodology used in the project for the research, design, implementation and testing. It also mentions the technologies used to achieve the goals.

1. Methodology

This sections is basically an extension to the project plan which had to be made during the first week of the project. An approximate guideline to follow the project was set focusing on the project deadlines. I left a few weeks for margin for error in case something takes slightly longer than planned for whatever reason.

——Project timeline——

For this project to be successful I am planning on spending the initial weeks researching relevant literature and becoming familiar with the concepts of Particle Swarm Optimisation. This is a completely new field to me and understanding the key ideas and models will be critically important. Not only will I need to understand PSO's background I will also need to study previous implementations and applications in order to become absolutely comfortable with it. Finally, as I am planning on improving an existing algorithm, I will have to spend some time becoming familiar enough with the code so that I will be able to modify it with ease.

The implementation stage will consist of designing the future system and the realisation of the plans. Key design decisions will have to be made during this stage and the solutions might be obtained from the analysis of previous work.

To complete this project test driven development will be carried out. I plan to test after every implementation or modification. This will be done to ensure that changes won't affect any previous functionality. The tests will evaluate the efficiency as well as the accuracy of my system. Given the nature of PSO's 'random' initialisation, I want to make sure that the results are consistent.

The writing of this result will be flexible, the sections will be written as needed or when the section arises naturally throughout the project.

2. Technology

2.1. Haskell. Coming from a strong mathematical background I find functional languages easier to understand. Also one huge advantage of pure functional languages is that the absence of side-effects allow them to offer a clear semantic framework to analyse the correctness of programs.

As Haskell is the functional language I am more familiar with, I didn't see the point in learning a new language as it would only restrict my project process, so Haskell was a clear winner.

There are other PSO implementations in other languages such as C and Ruby but as already mentioned, Haskell is my preferred language.

2.2. Operating System. As Haskell is platform independent (in the sense that it can be compiled in Windows, Linux or Mac) I have chosen to use Ubuntu 12.04 as it is my preferred OS and I feel the most comfortable with it. In addition, I wouldn't be affected in the about of software needed for the project as it is provided for all three OSs already mentioned.

The work was carried out on my personal laptop (Intel CORE^{TM} i3 @ 2.6GHZ,4Gb RAM). If required due to any reasons, the university provide classroom PCs (Intel CORE^{TM} i3-2100 CPU @3.10 GHz, 3 Gb RAM) although I have faith that my own machine will be reliable enough for me not to have to change machines. Sublime Text 2 was chosen as the IDE for the project. It has many useful functions [**14**] and similarly for the choice of OS, I am happy with this editor.

CHAPTER 7

System Design and Architecture

1. Original PSO Implementation

1.1. Initialisation.

1.2. Optimisation.

1.3. Termination.

2. Expansion for Portfolio Optimisation

2.1. Interface and User Input.

2.2. Optimisation.

2.3. Termination.

2.4. Fitness Function.

2.5. Presenting the Results.

CHAPTER 8

Financial Data

1. **Data Description**
2. **Problem Domain**
3. **Assets and their Weights**
4. **Analysis**
5. **PSO Parameters**
6. **Experimentation and Testing**
7. **Portfolio Constraints**
8. **Results**

CHAPTER 9

Experimentation and Testing

1. Constriction Factors

Maurice Clerc in his study on stability and convergence of PSO [**2**] has introduced the concept of constriction factor. Clerc indicated that the use of a constriction factor may be necessary to insure convergence of the PSO for certain fitness functions.

In order to ensure convergence of the PSO, the velocity of the constriction factor based approach can be expressed as follows:

$$V_i^{k+1} = K\left[V_i^k + c_1 r_1 \times \left(Pbest_i^k - X_i^k\right) + c_2 r_2 \times \left(Gbest^k - X_i^k\right)\right]$$

where (9.1)

$$K = \frac{2}{\mid 2 - \phi - \sqrt{\phi^2 - 4\phi} \mid} \text{ and } \phi = c_1 + c_2 \text{ s.t. } \phi > 4$$

The convergence characteristic of the system can be controlled by ϕ through choosing suitable c_1 and c_2 in (9.1). In this approach, ϕ must be greater than 4 to guarantee stability [**6**].

1.1. Testing Strategy. I plan to test whether this will in fact affect the outcome of the algorithm when applied to the portfolio selection problem, furthermore if it does affect it, then whether it improves or worsens the result. In order to test this I will conduct six different experiments, three without a constriction factor where one has the adjustment parameters taken from MEH Pefersen in his paper "Tuning & Simplifying Heuristical Optimization" [**11**], one with the two randomness coefficients that add up to less than 4 and one where they add up to more than 4. Then three more with a constriction factor and the rest is the same as the previous three.

All tests will be set to 50 particles, 500 iterations and 10 assets. This will give enough indication on how the constriction factor affects my fitness function and how it differs under various criteria.

1.2. Hypothesis. Constriction factor when applied to my portfolio selection problem with appropriate coefficients will improve the portfolio selection problem.

1.3. Results. This subsection shows the results and the following Table 1 contains the exact values for the results in my experiment, it will follow by a short explanations of the results.

Test	Mean Result	Standard deviation
WO-CF Pefersen	0.914099	1.61088×10^{-16}
WO-CF < 4	0.914099	1.72748×10^{-16}
WO-CF > 4	0.9141	5.36229×10^{-6}
W-CF Pefersen	0.91415	0.0000389374
W-CF < 4	0.91416	0.0000448388
W-CF > 4	0.914099	2.81328×10^{-16}

TABLE 1. Results for Constriction Factors.

Table 2 shows what the acronyms in Table 1 mean.

WO-CF Pefersen	:	Without constriction factor and Pefersen coefficients
WO-CF < 4	:	Without constriction factor and $\phi < 4$
WO-CF > 4	:	Without constriction factor and $\phi > 4$
W-CF Pefersen	:	With constriction factor and Pefersen coefficients
W-CF < 4	:	With constriction factor and $\phi < 4$
W-CF > 4	:	With constriction factor and $\phi > 4$

TABLE 2. Key for Results for Constriction Factors.

As stated in 9.1, $\phi > 4$, one can see from Table 1 shows this in W-CF Pefersen and W-CF < 4, as violating this constraints result in a serious decrease in the solution obtained.

What is interesting here is that PSO for my fitness function is efficient and consistent without the constriction factor as shown in Table 1 where WO-CF Pefersen and WO-CF < 4 both have the same mean and almost exact standard deviation, meaning they behave the same. Once we introduce the constriction factor, it is almost catastrophic if we don't have $\phi > 4$, as both W-CF Pefersen and W-CF < 4 have less efficient means and huge standard deviations (in comparison to the other tests) meaning they are unstable and unreliable. Once we make $\phi > 4$, the algorithm settles back to normal but does display higher standard deviation.

1.4. Conclusion. Constriction factor when applied to my portfolio selection problem does not improve the results and in the contrary to the hypothesis, it makes the results more unstable. The algorithm will therefore not include a constriction factor.

2. Scalability

Number of assets

3. Penalty value

For fitness function

4. Asset percentage/Induced/Forced Diversification

Constraint that says you have to invest between 0.05 to 0.35 on each asset

5. Risk and Risk Aversion

Level of riskiness

6. Efficiency

Standard Deviation stuff, box plots blahh blahhh

CHAPTER 10

Future Work

1. PSO Parameters

Dunno...

2. Self-termination

Stops after some criteria is met

3. Diversification

One of the most interesting concepts in portfolio theory I found was that of diversification, unfortunately I found this very late into my project and unable, due to time constraints, to include this into my application. Diversification excites me as it contradicts intuition. One would think that if one have one risky asset, adding another one would only increase the overall risk further, in fact it does the exact opposite!

The most simplistic model to represent this concept is the proverb, "putting ones eggs in more than one basket". Regardless on what the probability of each egg is, having more baskets with eggs is more likely to preserve more eggs that less baskets with the same amount of eggs. In other words, if one basket crashes, one still have the the other eggs which where in different baskets.

Now something more useful and even less intuitive is that if ones invests in more than one company within the same sector, for example split all ones money equally (for simplicity in example) and invest in all the mobile phone networks there are. Now company x gets into trouble for some reason which affects the stock market (fraud, IT, quality etc.) and the stock price for x begins to fall, ones will find that the price of stocks for all the other companies goes up. This is due to the investors and business which was with company x now deciding to opt out of that company and therefore bringing more investors and business to all the other companies in the market.

My application has a sense of diversification due to the extra constraint which I added late in the project as a emergency diversification solution. It states that one must invest between 5% and 35% on each asset to force diversification. This is vaux or brute intelligence though, what could be useful if the application gives a little extra preference if it knows that some assets belong to the same sector.

4. Asset's Covariance

Risk in terms of

5. Market Relationships

Gold vs Money

This would be brilliant!!

6. Real-time processing

Wow!

CHAPTER 11

Discussion and Conclusion

1. Discussion

2. Future Work

3. Conclusion

Bibliography

[1] W. Breen. Specific versus general models of portfolio selection. In *Oxford Economic Papers*, volume 36, pages 361–368, 1968.

[2] M. Clerc. The swarm and the queen: towards a deterministic and adaptive particle swarm optimization. In *Evolutionary Computation, 1999. CEC 99. Proceedings of the 1999 Congress on*, volume 3, pages –1957 Vol. 3, 1999.

[3] M. Clerc and J. Kennedy. The particle swarm - explosion, stability, and convergence in a multidimensional complex space. *Evolutionary Computation, IEEE Transactions on*, pages 58–73, Feb 2002.

[4] M. S. Fekdstein. Mean-variance analysis in the theory of liquidity preference of portfolio selection. In *Review of Economic Studies*, volume 36, pages 5–12, 1969.

[5] J. Kennedy and R. Eberhart. Particle swarm optimization. In *Neural Networks, 1995. Proceedings., IEEE International Conference on*, volume 4, pages 1942–1948 vol.4, Nov 1995.

[6] Shi Lim and Mohammad Montakhab. A constriction factor based particle swarm optimization for economic dispatch, 2009.

[7] Billio M. Simulation based methods for financial time series, 2002.

[8] H. Markowitz. Portfolio selection. In *Journal of Finance*, volume 1, pages 77–91, March 1952.

[9] H. Markowitz. Portfolio selection-efficient diversification of investments, 1959.

[10] K.E. Parsopoulos and M.N. Vrahatis. Recent approaches to global optimization problems through particle swarm optimization. *Evolutionary Computation, IEEE Transactions on*, 1:235–306, 2002.

[11] M.E.H. Pedersen. Tuning & simplifying heuristical optimization, 2010.

[12] Pablo Rabanal, Ismael Rodguez, and Fernando Rubio. `http://antares.sip.ucm.es/prabanal/english/heuristics_library.html`. Accessed: February,2014.

[13] Yuhui Shi and R. Eberhart. A modified particle swarm optimizer. In *Evolutionary Computation Proceedings, 1998. IEEE World Congress on Computational Intelligence.*, pages 69–73, May 1998.

[14] Jon Skinner. `http://www.sublimetext.com`. Accessed: February,2014.

[15] G Szego. Measure of risk. In *Journal of Operational Research*, volume 163, pages 5–19, 2005.