Heuristic Approaches to Portfolio Optimization

by

Abubakar Yahaya

BSc (Statistics)

Usmanu Danfodiyo University, Sokoto, Nigeria.

MSc (Decision Modelling & Information Systems),

Brunel University, UK.

This thesis is submitted for the award of degree of

Doctor of Philosophy

in the

Department of Management Science,

Lancaster University Management School

December 2010.

ProQuest LLC.
789 East Eisenhower Parkway
P.O. Box 1346
Ann Arbor, MI 48106 – 1346

Dedication

To

My lovely and caring wife: Amina

My beautiful and lovely daughter: Fātima.

My gorgeous sons: Muhammad, Ahmad and (my little Lancastrian) Al-Ameen.

Acknowledgement

All praises and thanks are due to Almighty Allah (TWT), the LORD of all the seven heavens and earths alike. He is the LORD of whatever is contained therein. I praise Him, thank Him and ask for His forgiveness over all my sins and shortcomings. He is the provider of all provisions and sustenance to all living things. It is in the Name of such Allah (TWT), I begin to write this thesis.

May the Peace, Blessings and Mercy of Allah (TWT) be with His most beloved slave, Prophet and Messenger, our Saviour and our Guide in person of Sayyidina Muhammad (SAW). May the same blessings be with the rest of Prophets and Messengers of Allah, pure (Ahl-Albayt) descendants of the Holy Prophet, his honourable companions and all those who follow the guidance of Islam until the Day of Resurrection.

I would like to acknowledge the assistance and support I received from my supervisor, Professor Mike Wright, who upon all his numerous commitments sacrificed most of his precious time in supervising my research progress over the whole of my three-year research journey. I really say a big "thank you" for all your patience and support. I would also like to thank my *viva voce* examiners who managed to arrange a suitable

time for my *viva*, and most especially Professor Richard Eglese who has been part of my examiners right from my upgrade up to my final *viva*. Thank you, Richard.

I would like to use this medium also to thank many, who positively touch my life in one way or the other. The first and foremost is my mum: Hajiya Khadeejatul Kubra whom I revere, love and admire more than any living mortal. She brought me up morally, guides me, prays for me, advices and put me on the right track. As for me, she really did her duty, and I'll continue to appreciate her efforts, generosity and love towards me up to the end of my life. Thank you, Mum.

My beloved, respectful, obedient, and caring wife: Amina Hassan Abubakar (Mrs) deserves all praises and thanks. Her advice, assistance, care, prayers and love helped a lot in shaping my life and our children's. I appreciate (and will continue to do so) the sacrifices she made during my three-year academic journey and the role she plays as a wife and as a mother to our children. I Love you because you are really great. Remain blessed, Darling!

I would like to thank a colleague and friend of mine: Dr. Amadou Gning (Department of Computing, Lancaster University, UK) whose invaluable criticisms and advices helped in shaping the composition of this thesis. He helped me a lot in my domestic and academic struggles. Amadou! May Allah (TWT) reward you immensely.

My brothers, sisters, in-laws, friends and well-wishers deserve a word of commendation for their constant support and prayers in actualization of my academic dreams; thank you all and May Allah (TWT) continue to help and guide us all.

I would also like to acknowledge the support I received from my friends at Lancaster University Islamic Society (LUISOC). Similarly, I would like to acknowledge the support and assistance of Miss Gay Bentinck (PhD Coordinator, Department of Management Science, Lancaster University, Lancaster); Thank you all.

Finally, I would like to express my heartfelt appreciation to the management of Petroleum Technology Development Fund (PTDF, Abuja) for funding my PhD programme.

Declaration of Authorship

I, Abubakar Yahaya, declare that this thesis entitled "Heuristic Approaches to Portfolio Optimization" is my own work and has not been submitted for the award of any similar or higher degree elsewhere.

List of Publications

(i) Yahaya, A. and Wright, M. B. (2009). *Effect of Initial Solution on Heuristics' Performance: A Perspective from Mean-Variance Portfolio Optimization Problem.* Presented at the 1st Student Conference in Operational Research (SCOR2009) held at Lancaster University, Lancaster, UK [March 27th - 29th, 2009].

(ii) Yahaya, A. and Wright, M. (2009). *On Portfolio Selection Using Metaheuristics.* Presented at the 10th Annual Meeting of the EUropean chapter on MEtaheuristics (EU/MEeting 2009) held at the Instituto Superior de Engenharia do Porto (ISEP), Porto, Portugal [April 29th – 30th, 2009]. (http://www.dcc.fc.up.pt/eume2009/pdf/proceedings.pdf).

(iii) Yahaya, A. and Wright, M. (2009). *Solving a Constrained Portfolio Selection Problem Using Particle Swarm Optimization.* Presented at the 3rd Annual Graduate Conference on Social Sciences and Management held at Norcroft Centre, University of Bradford, Bradford, UK [October 8th – 9th, 2009]. (http://www.bradford.ac.uk/lss/gradschool/conference/abstracts.pdf).

(iv) Yahaya, A. (2010). *Portfolio Selection Using Genetic Algorithms.* Presented at the 2nd Annual Postgraduate Research 'Creativity and Change' Conference held at Lancaster University, Lancaster, UK on 20th March, 2010. (http://createandchange.org.uk/)

(v) Yahaya, A. and Wright, M. (2010). *SWAN – A hybridized approach for solving Portfolio Selection Problems (PSP).* Presented at the 2nd Student Conference on Operational Research (SCOR2010) held at Sir Clive Granger Building, University Park campus, University of Nottingham, Nottingham, UK [April 9th – 11th, 2010]. (http://www.scor2010.co.uk/downloads/scor_proceedings.pdf).

Abstract

One of the most frequently studied areas in finance is the classical mean-variance portfolio selection model pioneered by Harry Markowitz; which is also, undoubtedly recognized as the foundation of modern portfolio theory. The model in its basic form deals with the selection of portfolio of assets such that a reasonable trade-off is achieved between the conflicting objectives of maximum possible return at a minimum risk, given that the right choice of constituent assets is made and proper weights are allocated. However, despite its enormous contribution to this branch of knowledge, the model is not immune from criticisms ranging from those associated with its in ability to capture the realism of an investment setting – such as transaction costs, cardinality constraints, floor and ceiling constraints, etc.

In this research we extended the classical model by incorporating into it the cardinality as well as the floor & ceiling constraints after which we implemented six different metaheuristic algorithms to solve this advanced model. We then designed and implemented some *neighbourhood transition strategies* to enable our designed algorithms solve the problem in an efficient and intelligent way.

Furthermore, we proposed a new portfolio selection model with target-semivariance (as defined in a previous research) as the objective, and constrained by additional real life (cardinality and floor & ceiling) constraints.

List of Tables

List of Figures

Table of Contents

1.0 Introduction

1.1 Overview of the Research

Investments decisions, especially in financial (capital) markets are thoughtfully reached, by individuals or fund managers (such as pension trustees, stock brokers, etc) who use savers' or other corporate entities' monies to purchase a single asset or bunch of assets with the sole motif of potentially maximizing their clienteles' future expected returns. However, any investment (and by extension, almost any type of financial transaction) has an element of risk and/or uncertainty attached to it. This is so, because the actual future outcome of the potential return involved in such a deal cannot be guaranteed. Therefore it can be understood that, one basic feature of investment opportunities is that their actual return cannot be stated with any precision, thereby making them uncertain or simply risky, and this brings to light the inability of any individual, fund manager, or any other third party to ascertain (with 100% confidence) what the return on his investment will be in the very near or far away future. Realistically, no investment with certain/guaranteed returns exist; however, treasury bills and bonds are most of the times classified under the category of *guaranteed-return* (riskless) investments; but the true situation is that, even if (at the end of the investment period) there is a certain rate of return for these types of investments, whenever any natural phenomenon strikes – which may consequently trigger some other uncertain phenomena (such as inflation); it will make their rate of return to deviate from the normal trend, and hence not certain anymore.

The concept of risk or uncertainty in this context does not apply to only when the dispersion (difference between the actual and expected return) is negative (downside risk), but also applicable when it is positive (upside risk) which is due to consequences

of "positive surprises or non-occurrences of some negative events" [100]. Fama [51] who started a still ongoing research on *information efficiency*, believes that when all the necessary information and expectations on future prices can be communicated by the current prices, then the future payoffs and returns can be regarded and treated as random numbers, this leads to the fact in the simplest case that the returns of an asset can be said to follow a normal distribution which can be characterized by the expected value (average or mean) of the returns and the variance (or standard deviation also known as the volatility) and which are believed to explain all the information about the expected outcome and the range of deviations from it. It is noteworthy at this juncture that, the valuations as well as returns on an asset are therefore, practically highly uncertain; this is because, had it been that these parameters were known with certainty, the investors' aim would be to set up a value maximization linear programming problem.

Many financial problems (including risk management, derivative pricing, asset allocation, model fitting and many more) which can be formulated as optimization problems are of immense critical importance; but undoubtedly, the most important among these classes of problems is the ***Portfolio Selection Problem (PSP)*** pioneered by Harry Markowitz [101, 103] whose main goal was to compute a portfolio of assets (from a set of available assets) with minimum risk (quantified by portfolio's variance) subject to achieving a given level of return. The model, apart from being one of the first mathematical frameworks providing investors with the tool to measure and quantify portfolio risk, is of immense importance as it suggests and justifies (analytically) ***portfolio diversification*** as a rational investment criterion, rather than paying much attention to maximizing return as the only parameter of interest. The

resultant theory is undisputedly regarded as the foundation stone of what is now known as ***Modern Portfolio Theory*** (**MPT**).

The breakthrough made by this theory within the global financial investment arena led to the recognition of the author's contribution to global financial practice, and this eventually resulted in the author's conferment with the award of the prestigious Nobel Prize (*Sveriges Riksbank Prize in Economic Sciences*) in 1990 together with M. H. Miller and W. F. Sharpe for their pioneering contributions in the theory of financial economics. Markowitz, in the early 1950s published his article in the *Journal of Finance* on portfolio construction strategies. The paper, entitled *Portfolio Selection* [101] has built the foundations of what is popularly referred to as ***mean-variance portfolio optimization***, ***mean-variance analysis*** and ***MPT***. It is widely believed that, the *mean-variance analysis* is highly influential in portfolio management practices. In its basic form, it provides a mathematical framework for selecting assets to form a portfolio based on their expected performance as well as the investors' risk tolerance.

However, despite the globally–acknowledged breakthrough brought by this optimization procedure more than half a century ago, it appears that this optimization procedure is mostly utilized at the more quantitative firms; while at many other firms, portfolio management remains a purely judgmental process based on qualitative and not quantitative assessments [50]. This is so, because, quantitative efforts in most of these firms seems to be directed at providing risk measures to investors and portfolio managers. These measures mostly help the portfolio managers/investors to assess and visualize the degree of risk involved in taking a particular portfolio, where it (risk) is defined as underperformance relative to a mandate. It should be noted here as well,

that the theory is a normative theory; in the sense that it describes a norm of behaviour that investors/portfolio managers should follow in constructing a portfolio of assets and this is in contrast to a theory that is actually being followed or adopted.

In our research, we intend to use some metaheuristic techniques to solve a constrained portfolio optimization problem with (Target) semivariance as an alternative to the Markowitz' conventional risk measure (Variance), while at the same time incorporating some real-world investment constraints – such as the cardinality and floor & ceiling constraints. These metaheuristics are, mostly, high-level techniques designed to guide some other (sub-ordinate) heuristics on a search space to find a very good solution to wide ranging optimization problems without necessarily guaranteeing optimality. They were found to be very successful as high-level criteria algorithms for solving hard combinatorial optimization problems arising from various Artificial Intelligence (AI) and Operational Research (OR) areas, such as the *Travelling Salesman Problem* (TSP) and *Constraint Satisfaction Problem* (CSP).

1.2 Motivation of the Research

In the field of Operations Research/Management Science (OR/MS), lots of problems believed to be of theoretical and practical importance are of a combinatorial nature. Combinatorial problems involve determining values for discrete variables such that some set of conditions/constraints are satisfied. These problems can be classed as either optimization or satisfaction problems. In the former, the main aim is to find an optimal configuration, ordering, grouping or selection of discrete objects usually finite in number [93]. The most notable example of these types of problems is the well-known TSP, in which the cardinal objective is to find the shortest (possible) route that

the *Travelling Salesman* – who is to visit all cities in his domain – follows, which at the end of the day will be found to minimize the distance covered by the salesperson subject to the condition (constraint) that each city must be visited once and only once before returning to his original point (city) of departure. Other examples include vehicle routing, assignment, facility location and scheduling problems. For satisfaction problems, a solution satisfying some restrictions/constraint has to be found. Graph colouring, frequency assignment and resource allocation problems are the most notable examples under this category.

It is sometimes easy to state a given combinatorial optimization problem, but finding a solution to it might be very difficult. For instance, when we consider a TSP, there exists no known algorithm that guarantees obtaining an optimal solution within a polynomial time domain. In the same vein, no algorithm can guarantee in a polynomial time, whether a given CSP is satisfiable or not. This type of phenomenon widely encountered in solving many (real life) combinatorial problems led to the emergence of an area of research popularly known as Complexity Theory [66, 5] – which aims at categorizing problems based on the degree of difficulty inherent in finding their solutions. Among these classes of problems, one (***NP-hard***) has a special property that: for any of its members, no algorithm exists to date that can solve the problem in polynomial time; and from the computational complexity perspective, if any of the *NP-hard* problems is to be solved by an algorithm in a polynomial time, then it is possible also for all problems in this complexity class to be solved in polynomial time. From computational point of view, these problems are *inherently intractable*; thus in the worst case scenario, *exponential run time* would be needed by any algorithm attempting to solve an *NP-hard* problem. CSP, TSP and likewise a

constrained PSP [84, 98] all belong to this class of problems and are therefore regarded among the most difficult combinatorial optimization problems.

It should be noted that, there are many real life problems that are combinatorial in nature upon which there is an urgent need for efficient algorithms. There are however, two classifications of algorithmic approaches for tackling combinatorial optimization problems, namely: the *exact* and the *approximate*.

In our research, we are considering an enhanced Markowitz model with two objectives, in which semivariance (as an alternative measure to portfolio risk) is minimized and portfolio expected return is maximized subject to achieving a given target; we also incorporate some of the practical constraints, namely: the cardinality and floor & ceiling constraints. This proposed model would, henceforth be referred to as ***Mean-Semivariance Portfolio Selection Model (ESPSM)***.

As in the case of the enhanced mean-variance (*E-V*) model (with constraints); our proposed ***ESPSM*** is also an *NP-hard* combinatorial optimization problem, due to the introduction of binary variables that handle the cardinality of a portfolio [84, 98]. For any NP-hard problem, the only viable option for obtaining a very good solution within a reasonable time frame is resorting to *approximate* algorithms such as Metaheuristics. Metaheuristics are certain classes of heuristic techniques which are found to be applicable to virtually all types of discrete optimization problems, and can also be adapted for use on continuous problems. These methods include Genetic Algorithms (GA), Simulated Annealing (SA), Ant Colony Optimization (ACO), Tabu Search (TS), Particle Swarm Optimization (PSO) and others [15].

As for this research, we seek to develop some solution algorithms to our proposed model by designing and implementing six different metaheuristic algorithms, among which two of them (SA and TS) are Local Search techniques, two (GA and PSO) are Evolutionary Algorithms (EA), one (Parallel SA) is a parallel implementation of SA and one (SWAN) is a hybrid of SA & PSO. All the aforementioned algorithms were designed and coded in C++ programming language (which we learned specifically to conduct this research).

1.3 Why it is important

When we consider the need for efficient algorithms capable of handling difficult optimization problems (both in academia and industry); we will come to the conclusion that our research will be of immense importance in that regard. For instance, our proposed (***ESPSM***) model when successfully implemented will serve as an alternative tool to investors/practitioners in optimizing their portfolios of assets especially as it incorporates the investor more-preferred risk measure (the semivariance) rather than the conventional variance measure. Similarly, the metaheuristic techniques designed will be found to be very useful in finding good solutions not only to most of the constrained formulations of the PSP within a practical reasonable time frame, but also to some other difficult global optimization problems, such as functions optimization (minimization/maximization).

1.4 Why it is difficult

The special nature (*NP-completeness*) of the problem makes it difficult for the conventional exact methods to arrive at an optimal solution in a reasonable time frame. Consequently, we had to design some metaheuristic algorithms to handle the situation.

When it comes to implementing the metaheuristic techniques developed, there are lots of issues that made this task extremely difficult. These involve reaching decisions that are peculiar to the metaheuristic algorithm in view and as well, issues involving decisions that are *problem-specific*.

One issue that makes this research difficult is in the implementation of the algorithms themselves to suit the nature of the PSP; as they need rigorous and very well focused *parameter fine tuning*. This is because a very slight change in a single parameter value may make the algorithm perform very (unexpectedly) poorly, and thus given the number of parameters a particular algorithm possesses, it is extremely difficult to achieve a set of parameters that work acceptably well.

Another challenge is in designing and developing our neighbourhood structure, since a good neighbourhood structure can immensely improve the efficiency and effectiveness of an optimization algorithm. In the preliminary design, we allowed a random *move* in generating a candidate (neighbouring) solution, but unfortunately this decision was found to be ineffective, in the sense that most of the solutions returned by the algorithms were found to be of inferior quality which may not be unconnected with the fact that the problem is continuous, while many of the techniques considered were originally designed for, and are usually used for discrete problems. So in order to improve our algorithms' performance we had to be a little bit creative by introducing a guided neighbourhood move which allows for *increasing* or *decreasing* a randomly chosen asset's weight from the current portfolio's configuration. Our neighbourhood definition also allows for the *transfer* of some fraction of an asset's weight to another;

and it occasionally considers *insertion* and *deletion* of new assets into and out of the current configuration.

Another unexpected difficulty was encountered in enforcing feasibility of a given candidate solution (in a constrained problem), especially after undergoing some processes involved in the neighbourhood move. For instance, when an asset weight is *decreased* by some specified (often known as *step*) value, and as a result it falls below the minimum threshold allowed, then the asset index together with its corresponding weight have to be *deleted* from the current portfolio's composition, as a result of which we were constantly being faced with a dilemma of how do we *insert* a new replacement in the portfolio and what would be the weight of the newly introduced asset; noting that, if the newly introduced asset's weight is too large, the resultant portfolio will be very different indeed from the current portfolio, and if it is set to take a very small value, the resultant solution will be prone to entrapment in a local optimum.

Going by the continuous nature of the PSP problem; in our TS implementation, we were also faced with another difficult challenge on how to declare a particular move as *tabu* (non-permissible). With the above in mind, we developed a new idea in which a newly generated (candidate) neighbouring solution is considered to be within the *tabu region* of the current, if the (Euclidean) distance between them does not exceed a specified threshold value.

We must admit at this point that, implementing the constrained PSP in PSO/SWAN was the topmost challenge encountered in this whole research. The PSO/SWAN is

very suitable for the unconstrained PSP problem; while for the constrained case, even if the algorithm managed to begin its search for the candidate solutions from a feasible region, the moment all the *particles* (candidate solutions) undergo a velocity update after which an eventual repositioning takes place, they would almost immediately lose their feasibility status and further repairs need to be done in order to regain feasibility.

1.5 Aims of the Research

The aims of this research include:

- Designing a portfolio selection model that will adopt semivariance as an alternative measure of portfolio risk.
- Designing a model that will reflect the actual real-life situation in the portfolio selection decision, by adding extra real and practical constraints such as the cardinality and buy-in threshold constraints.
- Solving some portfolio selection problems using some of the widely used metaheuristic algorithms, test their individual robustness and at the same time compare their respective performance in arriving at a very good solution.
- Solving some other difficult global optimization problems especially the continuous types, such as the De Jong's and Schwefel's functions.

1.6 Objectives of the Research

The main objectives of this research include:

- To provide decision makers (investors, fund managers and other stakeholders in financial investment) with the basic knowledge required to make an intelligent and sound decision in constructing portfolios from the pool of seemingly promising and non-promising assets, with the sole aim of

optimizing conflicting trade-offs; involving maximization of expected return while at the same time keeping risk as little as possible.

- To provide ideas and techniques of incorporating real life constraints obtainable in any asset management industry such as Buy-in threshold and Cardinality constraints and their resultant effect on the smooth curve of efficient portfolios, also known as "efficient frontier".
- To explore the robustness of the designed metaheuristic techniques in producing solutions of high-quality in a PSP problem whether or not such a problem is enriched by the so-called real life investment constraints.
- To test the capability of our designed algorithms in handling other difficult optimization problems.

1.7 Research Contribution

This research contributes in investigating and improving methods for getting good solutions to the enhanced Markowitz model while treating it as a 2-objective problem, and thus potentially enabling better solutions to be found.

One of the main contributions this research offers is the design of a newly developed (**SWAN** – **SW**arm **AN**nealing) algorithm, which is a hybrid of the PSO and SA. Although, we are fully aware of similar implementations in the research community, where PSO and SA are hybridized as in Wang and Li [147]; our implementation is significantly different. This is because, in Wang and Li [147] each of the generated *particles (candidate solutions)* underwent processes involved in the SA technique after which the best among them was declared the global best (*gbest*) solution, then other particles' positions were updated according to the PSO update mechanism, and the

process continued in similar fashion until convergence or a given stopping criterion was attained. In our implementation, however, all the generated *particles* undergo all the processes involved in the PSO technique until convergence, after which the global (*gbest*) solution is passed to SA as a starting solution which will then keep on being improved until a given stopping criterion is attained. By implementing the algorithm in this way; we hope to obtain solutions that are *at least* as good as those returned by either the PSO or the SA. This is because, in our implementation, we first aimed to exploit the PSO's *exploratory* capability by searching the entirety of the solution space to obtain a very good starting solution to be passed to SA, which will then be used to apply its *intensification* power (through successive decreasing temperature values) to obtain possibly a finer and better solution.

Another important contribution this research has on offer is the introduction of a new *neighbourhood* move structure geared towards guiding our Local Search methods (especially the SA, TS and parallel SA) to obtain not only good, but hopefully near optimal solutions. The move operations in the neighbourhood structure allows for *incrementing* and *decreasing* an asset's weight. It also allows for *deleting* an asset from a portfolio whose weight violates a vital constraint. This neighbourhood structure occasionally allows for the *insertion* of a new asset into the current portfolio with a negligible weight; it also, according to some probability value, *transfers* some portion of an asset's weight to another. We call our neighbourhood move structure **IDDIT** (**I**ncrease-**D**ecrease-**D**elete-**I**nsert-**T**ransfer) as a result of the operations that are randomly executed therein.

We also developed and implemented another neighbourhood structure for solving the constrained PSP problem using PSO and SWAN. This special (**IDDIT**-like) neighbourhood structure has some aspect of guidance enabling it to perform well with the algorithms it is designed for; as it allows for movement of *particles* (candidate solutions) in the search space without compromising solution quality. Although the particles use some information from their personal history and that of the entire swarm; they update their positions in the search space devoid of the conventional velocity and position update mechanisms, but rather, occasionally jump away from their immediate neighbourhood in search of a better solution while at the same time escaping entrapment in a local solution.

We also proposed a PSP model with target-semivariance (as defined in Estrada [47, 48, 49) as the objective (risk measure), while at the same time incorporating additional real life constraints including the cardinality and buy-in threshold (floor & ceiling) constraints.

1.8 Outline of the Thesis

This section is meant to briefly relate to the reader what is to be expected in the following chapters, sections and subsections in this thesis. The next chapter, after providing an overview of the classical Markowitz E-V model, it also defines what constitutes an E-V investor. Limitations and shortcomings of the classical E-V model as well as possible extensions of the classical model are also discussed. The chapter concludes by describing how a semivariance would be computed, after which follows a thorough explanation about the implementation of our proposed mean-semivariance portfolio selection model.

Chapter three, among other things, provides some definitions of metaheuristics as provided in the literature. A fairly detailed review of some literature on the successful application of metaheuristic algorithms in other areas of research, as well as in finance (portfolio optimization) is also reported. The chapter concludes with a fairly thorough explanation of the six metaheuristic algorithms used in this research.

Chapter four provides all the necessary details required for the solution of the PSP's unconstrained case. The chapter also discusses some performance and evaluation metrics which would be used to assess the performance of our algorithms amongst themselves as well as against a well known non-linear optimization solver (CPLEX). The chapter then concludes with discussion of the results obtained from the solver and our implemented algorithms.

The fifth chapter is concerned with the solution of the constrained PSP. The chapter contains a thorough explanation of the two neighbourhood move definition strategies developed in this research (which form integral parts of this research's contribution). The first one is called IDDIT and is meant to serve our local search algorithms; while the other (which is more advanced) is meant to serve our swarm algorithms. The chapter concludes by discussing the results and evaluation of the algorithms in relation to the constrained case outputs.

Chapter six is meant to assess how well our algorithms (PSO and SWAN) perform in solving some other optimization problems other than the PSP they were originally designed for. The chapter gives a very brief overview of some popular, but standard optimization test functions that are used to evaluate newly developed optimization

algorithms; these functions/problems include both constrained and unconstrained cases.

We draw some conclusions in chapter seven; and we also provide a hint on what we plan to do in order to take our research to the next level.

2.0 Portfolio Selection Strategies

2.1 Portfolio Selection: Classical Theory and Extensions

One of the most important aspects of asset management is the process of intelligently combining a set of attractive assets into a single *master* asset often called a portfolio of assets. Realistically, these portfolios are strongly required to be optimal as far as the trade-offs between the conflicting objectives of maximizing returns and minimizing risk are concerned. However, before the advent of MPT, portfolio selection decisions were purely hinged on qualitative assessments of the available assets, while the idea of incorporating real-life constraints into the process of assets' selection was almost nonexistent.

2.2 The Markowitz Mean-Variance model

The concept of MPT was pioneered for more than half a century by Harry Markowitz, hence the name Markowitz' Modern Portfolio Theory. Although, this modern concept is unarguably regarded as the *foundation stone* of modern day portfolio theories, it is simplistic and at the most basic level, in the sense that some of the underlying assumptions upon which it is based cannot be met practically, while at the same time turning a *blind eye* to the practical considerations of cardinality constraints (which restricts the number of assets that should be incorporated into the portfolio), minimum transaction lot sizes, transaction cost, liquidity constraints and much more. The most prominent and important assumption of this theory, is that, investors are basically *risk-averse*, a feature that measures the degree of investor's preference on his/her investment objectives. Literally, this means, if an investor is *pushed* to make a choice between two assets with similar expected returns, but with varying magnitude of (risk) variance, he/she would prefer the one with smaller variance (risk). Likewise, if he/she

was to be presented with, and asked to make a choice between another set of (two) assets with equal variance and different expected rate of returns, he/she would go for the one with higher expected return.

Another reason or evidence supporting the above claim is Markowitz's observation to the fact that investors generally hold diversified portfolios. He argued that if the reverse was the case, it would be observed that investors would always aim to hold a single asset that has the likelihood of bringing higher expected return irrespective of the risk involved in making such a decision. One more fact supporting the aforementioned claim by Markowitz is the purchase of different types of insurance (such as life, accident, health and automobile). Investors purchase insurance to avoid future uncertainties even if the premium (they would pay) is higher than the expected payoff of the insurance. Markowitz in the early 1950s, based on the presumption of risk-aversion behaviour of many investors proposed a (two objective optimization) model, in which: the expected return of an investment is maximized; and the risk (Variance of return) of investment is minimized

Markowitz' *E–V* model is normally regarded as the building block of the MPT [52]. It gives a multi-objective optimization problem, with two output dimensions. The model is based on the assumption that asset rate of returns exhibit the properties of a normal distribution, which means the distribution of the rate of returns can be solely explained or described by the first two moments (namely, expected value, E and variance, V) of the distribution. So, the mean of the asset rate of returns (E) can be used as the expected return in the long run period of time, while the variance of the asset rate of

returns (V) can be used to denote the degree of riskiness involved in holding such an asset in an investment decision; hence the term E–V model.

Suppose an investor is faced with a universe of n assets out of which he has to make some choices in order to make up a portfolio of investments. Furthermore, assume that the mean (or expected) rates of return of these n assets can be denoted by $\bar{r}_1, \bar{r}_2, \ldots, \bar{r}_n$ and the covariances are denoted by σ_{ij}, for all assets $i, j = 1, 2, \ldots, n$. Then, a portfolio P consisting of these n assets with fractions of weights often denoted by w_i is to be found, in which these fractions add up to unity. The observance of a negative weight means *short selling* is allowed in the portfolio P. One way of defining the problem facing a potential investor is that he should find a possible combination of assets (portfolio) with least (minimum) variance, with expected value of the portfolio fixed at some value R_P^*. Then a feasible portfolio with this minimum variance and expected value can be found by formulating and solving a Quadratic Programming (QP) problem as follows:

$$\text{Minimize} \quad \sigma_p^2 = \sum_{i=1}^{n}\sum_{j=1}^{n} w_i \sigma_{ij} w_j \qquad 2.2(a)(i)$$

Subject to

$$\sum_{i=1}^{n} w_i \bar{r}_i = R_p^* \qquad 2.2(a)(ii)$$

$$\sum_{i=1}^{n} w_i = 1 \qquad 2.2(a)(iii)$$

$$0 \le w_i \le 1 \qquad 2.2(a)(iv)$$

Where, $\bar{r}_i$ is the expected rate of return for asset; R_P^* is the desired return from the portfolio P; σ_{ij} is the covariance between asset i and asset j; σ_p^2 is the minimum value of portfolio risk at a given level of return; and w_i is the weight allocated to asset i.

The above formulation is a Convex Quadratic Program due to the fact that the program has a quadratic objective function, while the constraints (conditions) are made up of both linear equalities and inequalities.

When the problem is solved for a set of R_P^* values, the entire efficient frontier for the unconstrained problem can be estimated. It is now left for the investor to choose any efficient portfolio depending on his specific risk/return needs. The efficient frontier is composed of Pareto optimal portfolios, in which neither of the two criteria (risk and return) can be improved without deteriorating the other.

If the possibility of *short selling* assumption is dropped, which means the fractional weights of the selected assets must be non-negative ($w_i \geq 0$) for all assets $i = 1, 2, \ldots n$ then obtaining the finite maximum and minimum points on the variance-return (standard deviation-return) plane is easy and straightforward. This can be achieved by solving the Lagrangian formulation of the problem as follows:

Minimize $$(1-\lambda)\sum_{i=1}^{n}\sum_{j=1}^{n} w_i \sigma_{ij} w_j - \lambda\sum_{i=1}^{n} w_i \bar{r}_i \qquad 2.2(b)(i)$$

Subject to

$$\sum_{i=1}^{n} w_i = 1 \qquad 2.2(b)(ii)$$

$$0 \leq w_i \leq 1 \qquad 2.2(b)(iii)$$

$$0 \leq \lambda \leq 1 \qquad 2.2(b)(iv)$$

λ is a Lagrangian term and can take any real value within the interval [0, 1] to determine what is known as efficient (non-dominated) portfolios – those that can easily be seen to form a frontier (curve) of non-dominated portfolios on the Risk-Return plane. The Lagrangian term takes a value within the given interval based on the

investor's degree of risk aversion or tolerance. When λ takes the value 1, the objective function reduces to a linear function and thus the problem turns into a linear programming problem which seeks to maximize portfolio expected (mean) return irrespective of the degree of risk involved upon taking such a decision, and this type of decision may be taken by *risk–seeking* investors who end up sinking all their investment capital in only one single asset which seems to have the potential of a very high return in the future and at the same time might be the one with the highest risk.

On the other hand, when $\lambda = 0$, the objective function becomes a non-linear (quadratic) function thereby turning the entire problem into a Quadratic Programming (QP) model with the sole aim of minimizing the portfolio (variance) risk without taking the portfolio expected return into cognizance. Upholding such kinds of portfolio construction decision characterizes the investor as *strongly risk–averse* or *risk–hating*, in the sense that his/her main goal is to minimize investment risk at all cost not minding the meagre return he might end up with in the future. Figure 2.1 below shows a typical efficient frontier of portfolios.

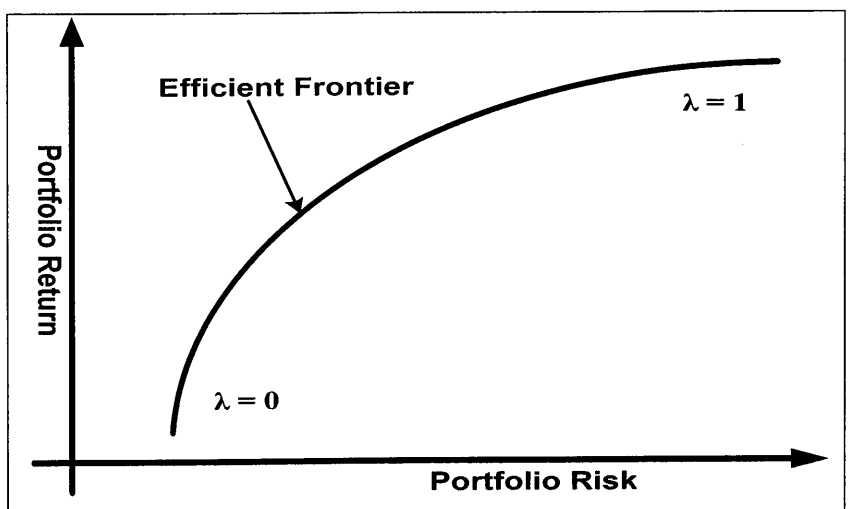


Figure 1: Typical unconstrained efficient frontier

The Markowitz problem provides the foundation for single-period investment theory. The problem explicitly addresses the trade-off between expected rate of return and variance of the rate of return in a portfolio. Once the Markowitz problem is formulated, it can be solved numerically to obtain a specific numerical solution.

Limitations/Drawbacks of Markowitz' Model

There are basically some limitations/drawbacks the original *E–V* model carries along with it, which consequently led to various criticisms thereby triggering further research in the area. The most notable among these criticisms are:

1. It is a single period model: This means once investors have made their decision concerning the allocation of wealth to different securities at the start of time period, they cannot take any further action until the next time period. This, in itself, is risky; because in the real life situation, portfolios are constructed such that they can be traded at any time. *Portfolio Rebalancing* strategies were proposed in Donohue and Yip [35], Jobst *et al* [84], and Calvet *et al* [16] to do away with this drawback.

2. The estimation of the underlying parameter inputs (return, variance and covariance) is considered to be another downside to this model. The calculation of mean, variance and covariance of returns are considered to be vital for accuracy reasons, as small errors can have large impact on the optimal asset weights. Konno and Yamazaki [91] proposed a computationally-less costly model without needing a covariance matrix as input.

3. Variance, as a measure of risk has been attacked over the last few years. Some investors/researchers including Konno and Yamazaki [91], Ballestero [7], and Harlow [67] view variance as a false indication of true risk. Since investors dislike negative deviation and embrace positive deviation from the mean, some argue that semivariance may be a better measure of risk (including Markowitz himself – see Markowitz [103] for details).

4. To obtain accurate results a large data set is needed. With this model being quadratic, the amount of time needed to solve for a large portfolio may also be impractical.

5. The model failed to consider realistic investment constraints obtainable in financial investment arena. These include, but are not limited to, floor & ceiling constraints, cardinality, turnover constraint, and transaction costs. Lots of researches were, and are still currently conducted to incorporate as many realistic constraints as possible into the portfolio selection model. See for instance Jobst *et al* [84], Chang *et al* [20], Crama and Schyns [25], Speranza [134], Hamza and Janssen [68], Bienstock [10], and Lee and Mitchell [94].

2.3 Extensions of the Classical model: Objectives

In the pre-modern portfolio era; investors, portfolio managers, pension fund administrators, and other stakeholders in the financial investment industry are very much aware of the existence of investments' return and risk. This means, they knew (positive) return is desirable and is what pushes the momentum of investments; while on the other hand, they believe there exists a risk (the undesirable component) attached

to realizing any perceived or expected return; furthermore, the greater their *feeling* of risk, the higher their *feeling* of uncertainty about realization of the expected return. However, despite their skills, long term experiences and knowledge of market behaviour and eventualities, the lack of any numerical measure to quantify risk of an investment remained the most disturbing challenge of the time.

As already mentioned, a new page in investment science was opened when Harry Markowitz developed in his seminal paper [101], a mathematical programming model for optimal portfolio selection. In the then newly developed model – based on multivariate normality assumption of the asset returns – Markowitz showed that the (desirable) portfolio return can be characterized and quantified by the first moment (Mean), while the portfolio risk would be characterized and quantified by the second moment of the distribution, also known as the variance. Since then, portfolio management and optimization techniques have developed immensely and variance became the most popular mathematical definition of (investment / portfolio) risk [81]. The resultant model is what would later be termed as the *E-V* paradigm of portfolio selection and whose wider acceptability as the bedrock of modern portfolio theory can never be overemphasized [7].

The Markowitz' *E-V* model is composed of three main features, namely the *objective*, *constraints* and *variables*. But most of the researches conducted in portfolio selection result from modifications made to one or more of the above mentioned features. Due to the theory's stand on upholding the multivariate normality assumption of asset returns, which eventually leads to making use of variance as a measure for quantifying portfolio risk, the newly found theory was welcomed with various criticisms.

Researchers who are opposed to employing variance as an appropriate measure for portfolio risk have suggested alternatives. Ballestero [7], Konno and Yamazaki [91], Roy [127], Huang [81], Feiring *et al* [53], Hamza and Janssen [68] were among the few researchers who fall in this category. According to them, the normality assumption of asset returns is not realistic, as asset returns are known to exhibit a high degree of asymmetry, also known as kurtosis in Statistics. Furthermore, they argued variance imposes a penalty on both positive and negative fluctuations relative to the portfolio expected return. Their argument is based on the fact that the majority of investors (especially Risk Averse), would normally be happy with the positive fluctuations, while at the same time being unhappy with any return below their expected or target return; after all, one of the main aim of investment is to gain a positive return; and thus there is no basis or justification whatsoever to penalize it and consider its contribution as an addition to the magnitude of asset's or portfolio's risk – which is what variance does.

In order to prove the preference and efficiency of downside risk over the classical *E-V* optimization framework; Feiring *et al* [53] sampled ten years data of monthly returns for 60 Hong Kong stocks in order to make different portfolios for the purpose of comparison. The model proposed, needed neither the normality assumption of the asset returns nor the estimation of covariance matrix – which are both, vital in the classical *E-V* paradigm – yet it was shown to have the potential of outperforming the latter method in various runs of the algorithms. The study also found out "there is a tendency that the longer the holding period, the higher the portfolio realized return", and this is not unconnected with the fact that: holding portfolios for a longer term

results in very few transactions, which would consequently lead to paying low transaction costs.

In another development, Ballestero [7] proposed a downside risk model (using Mean–Semivariance approach) as a substitute to the conventional Mean–Variance paradigm. Although (like the Markowitz model) computationally demanding, the risk measure used is believed to be a better representative metric of portfolio risk. The model was solely based on the validity of the beta regression equation proposed by William Sharpe. The study provided an insightful and illustrative numerical example (using fictitious data), on how to handle the multitude of computations involved therein. The newly proposed model was found to *beat* its Mean–Variance counterpart in 5 of the 7 scenarios generated. At some coincidental (similar) solutions, it is observable that the model has an edge over its mean – variance counterpart; in the sense that, it allows an investor to select a portfolio that can satisfy three interesting constraints namely: (i) expected return to his (investor's) target, (ii) minimum variance (risk), and (iii) minimum downside risk below the mean value. The research concluded by giving some credits to the newly proposed model, especially in the computation of semivariance matrix and its evident robustness, looking at how the model can be possibly extended to "convert its objective function into a downside risk measure for returns below some target or specific threshold other than the mean value", and/or to include some more relevant factors other than the market (single) one in estimating an asset return.

Another research that viewed optimizing portfolios from the mean-semivariance (*E-S*) perspective was conducted by Hamza and Janssen [68], and incorporating additional

realistic constraints namely: transaction costs, minimum transaction units and investor's portfolio holding. Although, neither was supported by any numerical example nor was it compared to any other benchmark, the research claims to be able to solve real life mixed integer programming problems in a short computational time when the model is incorporated in a suitable heuristic method. Another credit to the proposed model is the fact that it was not based upon any "probabilistic assumption on the distribution of stock data in the market", and in situations where the rates of return exhibit a multivariate normal distribution behaviour, they claim that it can be shown to be equivalent to the Markowitz model (though they do not prove this).

Konno and Yamazaki [91] argued that, there exists an immense computational challenge involved in estimating the parameters needed to optimize a Mean–Variance portfolio; in the sense that it is necessary to compute or estimate all the elements of the dense covariance matrix, and optimizing a quadratic programming problem with such a huge number of estimated elements embedded in the objective function requires an exponential period of time. The above challenges motivated Konno and Yamazaki [91] to design and propose a linear programming optimization model which is believed to improve the theoretical framework while at the same time reduce the computational burden inherent in the Markowitz model. The newly proposed model was not only proven to be equivalent to, and computationally advantaged over the former, but was also shown to be easier to update while at the same time not increasing the number of functional constraints irrespective of the number of stocks included in the model.

Depending upon how financial investments decision makers (investors, pension fund and portfolio managers) interpret or view the concept of risk; it is widely believed that metrics used in quantifying portfolio risk are categorized into two classes. The first classification includes the so-called *symmetric (two-sided) measures* which seek to penalize both positive (profit) and negative (loss) dispersion from a pre-specified value. The most common risk measures under this category are the Mean Absolute Deviation (MAD) as applied by Konno and Yamazaki [91] and Atkinson [6], as well as the popular variance or standard deviation as pioneered by Markowitz [101, 103].

The other category involves those metrics that aim to quantify risk subject to results and probabilities below some specified values, and these are normally called the *asymmetric measures* of risk. Notable ones among this category are the Semivariance as proposed again by Markowitz [102], safety first criterion by Roy [127], A risk curve metric (generalization of Roy's Safety first) by Huang [81], Value at Risk (VaR) by Morgan [114] as well as its extension – Conditional VaR (*CVaR*) by Uryasev and Rockafellar [145] and importantly the Fishburn's α–t criterion [56] which serves as a generalization not only for the asymmetric measures listed above, but also to their symmetric counterparts.

2.4 Extensions of the Classical model: Constraints

The classical Markowitz model can be regarded as the most basic formulation of the portfolio optimization problem. The enhanced version of the Markowitz model is basically the conventional model enriched by some realistic constraints, which makes it difficult for the well known exact algorithms to find solutions easily or within a

reasonable time frame. The most notable constraints, some captured and others not, by the original model include:

2.4.1 The basic (Return and Budget) Constraints

Budget and return constraints are the most important set of constraints in PSP. The budget constraint states that assets' weights must sum up to unity; while the return constraint ensures that the weighted sum of asset returns must be strictly equal (or ≥) to some target return. These constraints were included in the Markowitz original model playing an important role in determining the feasibility of a given solution. They take the form:

$$\sum_{i=1}^{n} w_i = 1 \qquad 2.4.1(a)$$

$$\sum_{i=1}^{n} w_i \bar{r}_i = R_T \qquad 2.4.1(b)$$

Constraint 2.4.1(a) ensures that all the investment capital is fully invested; while constraint 2.4.1(b) ensures that the portfolio return achieves a given target, R_T.

2.4.2 The Floor & Ceiling constraints

In PSP, *floor & ceiling* (often regarded collectively as *buy-in threshold* or simply *threshold*) constraints are very important. They are introduced into the mean-variance model to reduce much administrative costs resulting from holding an asset with negligible contribution towards portfolio's expected turn-over and performance, and/or to avoid over dependence upon one of the constituents (assets) chosen to make up the portfolio.

A *floor* (i.e. lower bound) constraint is a constraint that imposes a restriction on the minimum proportion allowed to be held by any given asset that forms part of a given portfolio. On the other hand, a *ceiling* (i.e. upper bound) constraint imposes a restriction on the maximum proportion any given asset is allowed to have (when it forms part of a given portfolio).

Floor & ceiling constraints are un-avoidably needed to optimize real world portfolio optimization problems and they can be respectively denoted by say a lower limit l_i and upper limit u_i. These constraints are formulated using a discrete programming modelling structure; which is well known using variable upper and lower bounds or semi-continuous variables. Using the finite bounds l_i, *and* u_i for the stock weight w_i, it can easily be comprehended that the following relationship holds: $l_i \leq w_i \leq u_i$. The introduction of a decision variable δ makes the formulation easier and straightforward, it should be noted here that our decision variable δ is at the same time a binary variable taking a value 1 if asset i is included in the portfolio, otherwise it is forced to take a value 0. Now the corresponding buy-in threshold restriction can be represented by the constraint pair:

$$l_i \delta_i \leq w_i \leq u_i \delta_i \qquad 2.4.2(a)$$

$$\delta_i = 0,1 \qquad 2.4.2(b)$$

$$\forall i = 1, 2, ..., n$$

The above restrictions means that, the binary variable δ would be forced to take a value 1 if any asset i is held, thereby forcing the fraction w_i related to asset i to lie between lower l_i and upper bounds u_i respectively. Similarly, if asset i is not held, δ equals 0 and consequently, w_i takes the value 0. The introduction of the binary

variables transforms the quadratic programming (QP) to a quadratic mixed integer program (QMIP) which becomes larger in size and computationally challenging [84, 111].

This constraint is strongly related to the cardinality constraint (to be discussed next), as it can implicitly define the range of the cardinality constraint. For example, setting a lower bound of say $l_i = 0.2$ for each asset implicitly defines a maximum cardinality of 5 assets in a portfolio; so it is extremely important to ensure consistency between the two constraints when formulating a constrained case of PSP.

2.4.3 The Cardinality constraints

In a typical financial market setting, an investor or fund manager will be confronted with a very large number of different types of stocks from which to choose. According to the Markowitz' *E-V* model, if it would be possible to get hold of all the available stocks in the market, then the resulting portfolio would have been highly diversified and consequently less risky. But realistically taking up such a decision is very costly, thereby rendering such a decision highly inefficient. Thus an investor or fund manager may wish to limit the number of assets he/she owns or manages.

It is possible that, there exist some rules enacted by the financial markets' regulatory authority limiting investors from holding too many assets in their portfolio. On the other hand, investors may want to monitor the performance of individual assets or possibly wish to minimize (high) transaction costs (resulting from holding large number of assets). Basically, in order to do so, investors would be forced to reduce the number of assets in their portfolios, and this can be achieved by introducing

cardinality constraints in the model which limits the number of binary variables, already introduced in the preceding *Floor & Ceiling* constraints formulation to force the portfolio to have a fixed number of assets, say K, and this can be done by introducing the constraint $\sum_{i=1}^{n} \delta_i = K$, in the earlier formulation which ensures that only K of the total n assets make the portfolio's composition.

Jobst *et al* [84] argues that the *cardinality* constraint is intrinsically related to the *Floor & Ceiling* constraint, in the sense that, the higher the threshold limit the more it tends to restrict the number of assets in a portfolio. However, imposing *cardinality* constraints only (with no threshold at all) may lead to some very small non-zero asset weights [50]. Therefore it is best to include both constraints in the same portfolio optimization model.

2.4.4 Transaction roundlots restrictions

In a typical investment setting, assets are traded in discrete number of basic units of investment often known as *roundlots*. Investors are always required to make transactions in multiples of these *roundlots;* this, according to Jobst *et al* [84], tackles "the assumption of the infinite divisibility of assets inherent in the M-V rule".

Transaction *roundlots* are often expressed in fractional form, say f_i, of the investment capital; after which asset weights w_i are defined in relation to f_i and an integer number of *roundlots*, n_i. Thus, we may now have: $w_i = f_i \times n_i$, $i = 1, 2, \ldots N$. However, it should be understood at this point that, incorporating the *roundlots* constraints will make satisfying the budget constraint almost impossible. With this in mind, there is

the need to relax the budget constraint by introducing some (*undershoot*, ε^- and *overshoot*, ε^+) variables which would eventually be penalized with a very high cost, say M, in the objective function. With this new transformation, equations 2.4.1(a), 2.4.1(b) and 2.4.2(a) will respectively look like:

$$\sum_{i=1}^{N} f_i n_i + \varepsilon^- - \varepsilon^+ = 1 \qquad 2.4.4(a)$$

$$\sum_{i=1}^{N} f_i n_i \overline{r_i} = R_T \qquad 2.4.4(b)$$

$$l_i \delta_i \le f_i n_i \le u_i \delta_i \qquad 2.4.4(c)$$

$$n_i \quad \text{integer} \qquad \forall i = 1, \ldots, N$$

$$\varepsilon^-,\ \varepsilon^+ \ge 0$$

2.4.5 Turnover and trading constraints

Although, Perold [119] was the first to implement minimum trading size constraints in his PSP formulation, it was Crama and Schyns [25] who elaborated more on it, and went ahead an extra mile to additionally incorporate turnover constraints.

Turnover constraints are responsible for imposing upper bounds on the variations of asset's holding from one time period to the next; while trading constraints impose lower bounds on such variations. For instance, if we denote by $w_i^{(0)}$, $\overline{P_i}$, $\underline{P_i}$, $\overline{S_i}$ *and* $\underline{S_i}$ the weight of asset i in the initial portfolio configuration, maximum purchase, minimum purchase, maximum sale and minimum sale bounds respectively; we can represent the turnover and trading constraints by:

$$\max\left(w_i - w_i^{(0)}, 0\right) \le \overline{P_i} \qquad 1 \le i \le n \qquad 2.4.5(a)$$

$$\max\left(w_i^{(0)} - w_i, 0\right) \le \overline{S_i} \qquad 1 \le i \le n \qquad 2.4.5(b)$$

$$w_i = w_i^{(0)} \ \text{or} \ w_i \ge \left(w_i^{(0)} + \underline{P_i}\right) \ \text{or} \ w_i \le \left(w_i^{(0)} - \underline{S_i}\right) \qquad 1 \le i \le n \qquad 2.4.5(c)$$

The first two set of constraints 2.4.5(a) and 2.4.5(b) represent the (turnover) purchase and sale constraints respectively, while equation 2.4.5(c) represents the trading constraint. According to Crama and Schyns [25], the trading constraint typically reflects the investor's inability or undesirability *to modify the portfolio by buying or selling tiny quantities of assets* due the existence of probably high fixed transaction costs or some contract clauses.

The apparent disjunctive nature of the trading constraint in 2.4.5(c) means: for any asset *i*, it is either the weight remains unchanged, or a minimum quantity $\underline{P_i}$ must be purchased, or a minimum quantity $\underline{S_i}$ must be sold.

2.4.6 Compulsory Constraints

Sometimes an investor may want his portfolio to contain specific asset(s) in a fixed proportion. Handling this type of constraint is easily done by fixing the value of the corresponding binary variable to unity. For instance, if an investor wishes that an asset with index, $i = 5$ must form part of his portfolio, then this can be achieved by setting $\delta_5 = 1$ as part of the constraints in a PSP formulation.

2.4.7 Class Constraints

Imposing class constraints in PSP formulation, although important and practically sensible, is rarely implemented in academic research [64]. It is possible an investor may want to compartmentalize the universe of assets into mutually exclusive groups (classes), each consisting of assets with similar attributes (Oil & Gas assets, IT assets, insurance, etc); after which he may limit the proportion of the investment fund to be allocated to each class.

For instance, let **H** be the set of classes, while L_k and U_k are respectively the lower and upper proportion limits for class *k*. Now we can define class constraint by:

$$L_k \leq \sum_{j \in k} w_j \leq U_k, \quad k = 1, \ldots, \mathrm{H} \qquad 2.4.7$$

This constraint can be used to diversify the portfolio across several economic divides.

2.4.8 Non-negativity bounds

The majority of the researches (with a few exceptions [25, 126]) conducted on PSP incorporate this constraint. It is defined by imposing the restriction:

$$w_i \geq 0, \quad \forall i \qquad 2.4.8$$

This basically means no *short sales* are allowed. It should be understood that this constraint will be rendered redundant by incorporating the *floor & ceiling* constraint in the PSP formulation.

The incorporation of one or more of the above computational constraints (with the exception of those described in sections 2.4.1 and 2.4.8) in the PSP transforms the quadratic programming (QP) to a quadratic mixed integer program (QMIP) which becomes larger in size and computationally challenging [84, 111], thereby making it much more difficult or even (most of the time) impossible to solve by the conventional exact methods embedded in most of the state-of-the-art nonlinear/quadratic optimization solvers (such as CPLEX, FortMP, MINOS and many more).

2.5 The Semivariance

The MPT was founded on the premise that all investment decisions are taken in order to achieve a return-risk tradeoff that is optimal in the opportunity set to some extent. However, in order to achieve this desired objective, an investor/portfolio manager will have to initially, evaluate the necessary information by quantifying *ex ante* measures of both risk and expected return for the appropriate set of assets. After that has been done, a set of *efficient* combinations of assets (providing the minimum risk subject to achieving a desired level of expected return) are isolated; upon which an investor/portfolio manager would choose a combination that is consistent with his/her risk tolerance level.

In this section, we intend to discuss one attractive, alternative-to-variance asymmetric measure of risk and more *investor-preferred*, that focuses on the returns below a specified target or benchmark return level (the semivariance), which will in turn be used to replace the classical Markowitz' risk measure (variance); further to that, we then enrich the resultant model with realistic investment constraints – specifically the cardinality and floor & ceiling constraints.

Although, there is a very clear way of identifying a portfolio of assets characterized by risk and return, the universally-accepted definition of risk is almost nonexistent or at least ambiguous [67]. This is because an investment decision perceived to be risky by one investor might not be viewed as such by another investor faced with similar investment scenarios and decisions during the same time period. For instance, one investor might consider risk as the probability of shortfall below some level of return; while another will be more concerned about the overall magnitude of loss, if any

should occur in the investment period. These different perceptions of the notion of risk and many other possible definitions remind us that variance (standard deviation) which is the conventional measure of risk, is deficient in dealing with rich set of portfolio objectives and constraints that investors/portfolio managers often formulate.

It should be made known that, there are several techniques developed over the years purposely for implementing the theory of portfolio selection; among which are the popular downside risk measures. The semivariance, however, is the most popular and commonly used of these set of measures. Moreover, it has been in use in many portfolio theory researches as long as the variance itself [115].

Roy's 1952 article [127], whose primary concept was that an investor should prefer to (first) safeguard his principal when dealing with investment risk, was extremely vital in the development of downside risk measures; this is because the tool he introduced and termed as *reward-to-variability ratio* allows investors to minimize the chances of their portfolios falling below a certain disaster level. Even Markowitz [103] acknowledged the strength of such an idea by admitting that investors will be interested in a downside risk (like semivariance), especially that the return distribution may not be Gaussian. Markowitz also showed that when the return distribution is Gaussian both downside risk and variance (his adopted risk measure) would provide a correct measure; while on the other hand, if the return distribution is non-Gaussian (asymmetric), only the downside risk measures would provide a correct answer.

Apart from the Roy's [127] and Markowitz' [103] aforementioned stand on the portfolio selection decisions based on downside risk measures in general, and

semivariance in particular, further research in proving the superiority of semivariance over the variance in portfolio selection context continued in the early 1960's and 1970's (see Quirk and Saposnik [120] and Mao [99] for further details). In the same vein, Roy [127], Markowitz [103] and Mao [99] all argued that: investors are not worried or concerned with the above-target returns but rather with the below-target returns and that semivariance is more consistent with financial and investment managers' perception of risk.

So, going by the above insightful revelations; the question that first comes to one's mind is: if such was the case then, why is the idea of adopting variance as a measure of portfolio risk received much more attention and preference by practitioners and academics alike? Partly in answering this question, Markowitz [103] argued that, variance is preferred because it has an edge over semivariance "with respect to *cost*, *convenience*, and *familiarity*"; so when he focused his attention on optimizing portfolios with variance as a measure of risk, other practitioners and academics followed suit and the rest is history.

We too, as Estrada [47] rightly argued, believe that the issue of *familiarity* should not preclude the use of semivariance, as this concept is wearing away over time, going by the fact that, downside risk portfolio analysis has increasingly been gaining attention and applied in both industry and academia as there are many downside risk measures [84, 47, 48, 49, 68] that are well known and widely applied.

In relation to the variance's advantages in *cost* and *convenience* over the semivariance; Markowitz [103] argued that [back then] "... *roughly two to four times as much*

computing time is required (on a high speed electronic computer) to derive efficient sets based on S_E [semivariance] than is required to derive efficient sets based on V [variance]". Furthermore, he added "*In an analysis based on V, only means, variances, and covariances must be supplied as inputs; whereas an analysis based on S requires the entire joint distribution of return*". While as far as convenience is concerned, Markowitz argued that, "*Unlike semi-variance, variance and standard deviation are known by many people acquainted with modern statistics*". However, as time went by, it can easily be observed that, all the above mentioned concerns have become much less an issue. For instance, a Forbes' article: Clash [22] stated that semivariance is already being used by many pension managers and is still gaining acceptability in some fund companies. In a similar development, some major funds use semivariance in calculating their risk-adjusted returns while at the same time including "*a relative measure of each fund's semivariance in annual and semiannual reports*". Other researches such as Estrada [45] show that, the cross-section inherent in the US and emerging market stocks can be explained by using semideviation and other downside risk measures. Estrada [47] also argues that portfolio managers and investors – especially pension fund managers due to their well known stand on preservation of principal and potential loss minimization strategies – should find downside risk measures (such as semivariance) immensely useful.

Semivariance, as already described above, is a risk concept that is believed to be consistent with both investors and portfolio managers' intuitive feeling of risk characterized by the failure to earn some target return. The *E-S* model allows decision makers to quantify risk from an arbitrary point rather than the mean value of the return distribution and also demarcate positive from negative deviations. One of the

advantages of this measure of portfolio risk is that, it allows investors to be more conservative towards losses (returns below some target), while at the same time becoming aggressive toward gains (returns above some target).

The semivariance, according to Harlow [67] and Markowitz [104], is defined as *an asymmetric measure of risk that focuses on squared return deviations below the mean of the distribution.* But target semivariance is similar and more general, in the sense that it considers return dispersions below any arbitrary target or benchmark return level. With this, it is important to note that, semivariance unlike the variance, does not increase with higher positive dispersions from the mean/target return, as these are rather captured by the mean of the return distribution.

If we assume an asset i has return r_{it} that are indexed over time t, the mean of asset returns can be computed using:

$$\mu_i = \frac{1}{N}\sum_{i=1}^{N}(r_{it}) \qquad 2.5(a)$$

The variance of returns for this asset can be computed by:

$$\sigma_i^2 = \mathrm{E}\left[(r_i - \mu_i)^2\right] = \frac{1}{N}\sum_{i=1}^{N}(r_{it} - \mu_i)^2 \qquad 2.5(b)$$

The covariance between any two assets i and j can be computed using the following expression:

$$\sigma_{ij} = \mathrm{E}\left[(r_i - \mu_i)(r_j - \mu_j)\right] = \frac{1}{N}\sum_{i=1}^{N}(r_{it} - \mu_i)(r_{jt} - \mu_j) \qquad 2.5(c)$$

Where μ_i and μ_j stand for the expected returns for both assets i and j respectively, while N represents the total number of observations.

Whereas the semivariance of asset i with respect to any given benchmark, B, according to Markowitz [103] is given by:

$$S_{iB}^2 = \frac{1}{N}\sum_{i=1}^{T}\left[\left(r_{it} - B\right)^2\right] \qquad 2.5(d)$$

And the semicovariance of 2 assets i and j with respect to the same benchmark value, B, can be computed by the following expression:

$$S_{ijB} = \frac{1}{N}\sum_{i=1}^{T}\left[\left(r_{it} - B\right)\times\left(r_{jt} - B\right)\right] \qquad 2.5(e)$$

Where the summation is only over the T time periods in which an asset return underperforms the benchmark, B.

The above definition as Estrada [47, 48, 49] observed, has an advantage and one disadvantage alike. The positive side of this approach is that, "it provides an exact estimation of the portfolio semivariance". On the other hand, the negative side is that, the semicovariance matrix is endogenous, which simply means when such semicovariance values of any two assets are incorporated into the computation of portfolio semivariance (in an *E-S* Optimization framework) any "change in weights affects the periods in which the portfolio underperforms the benchmark, which in turn affects the elements of the semicovariance matrix".

In another research, Hogan and Warren [77] attempted to estimate semicovariance between any 2 assets i and j by the following expression:

$$S_{ij}^{HW} = \mathrm{E}\left\{\left[\left(r_i - R_f\right) \times Min\left(r_j - R_f, 0\right)\right]\right\} \qquad 2.5(f)$$

Where R_f signifies a risk-free return and the superscript HW denotes that this definition was proposed by Hogan and Warren.

Similarly, the above definition too, Estrada [49] argues, has two main disadvantages:

(1) The benchmark return value is fixed to the risk-free rate, R_f thereby making it impossible to use any other different benchmark value.

(2) The semicovariance matrix is usually asymmetric (since it can be shown that $S_{ij}^{HW} \neq S_{ji}^{HW}$). This feature is particularly more limiting, as intuitively it is extremely difficult if not impossible to clearly interpret the contributions of both assets i and j to the portfolio's risk.

Thus, in order to address the above mentioned drawbacks; Estrada [47, 49] proposed an approximate expression to evaluate semivariance of asset i with respect to any given benchmark, B; given by:

$$S_{iB}^2 = \mathrm{E}\left\{\left[Min\left(r_i - B, 0\right)\right]^2\right\} = \frac{1}{N}\sum_{i=1}^{N}\left[Min\left(r_{it} - B, 0\right)\right]^2 \qquad 2.5(g)$$

and

$$S_{ijB} = \mathrm{E}\left\{\left[Min\left(r_i - B, 0\right) \times Min\left(r_j - B, 0\right)\right]\right\} = \frac{1}{N}\sum_{i=1}^{N}\left[Min\left(r_{it} - B, 0\right) \times Min\left(r_{jt} - B, 0\right)\right] \qquad 2.5(h)$$

for the computation of semicovariance between any two assets i and j.

The main advantage of these estimates is that, they can be used with any desired benchmark return value, B, and at the same time generate a symmetric ($S_{ijB} = S_{jiB}$)

exogenous semicovariance matrix; as both the symmetry and the exogeneity of this matrix are very critical tools for the implementation of the proposed model.

Going by the provisions above, Estrada [49] proposed computation of portfolio semivariance with respect to a benchmark, B, by:

$$\Theta_{PB}^2 = \sum_{i=1}^{n}\sum_{i=1}^{n} w_i w_j S_{ijB} \qquad 2.5(i)$$

This looks very much similar and behaves the same way as the portfolio variance given in equation 2.2(a)(i):

$$\sigma_p^2 = \sum_{i=1}^{n}\sum_{j=1}^{n} w_i \sigma_{ij} w_j$$

2.6 The Proposed (enhanced) Model

In this section, we intend to discuss the reasons why we decided to adopt (target) semivariance as the objective in our model for optimizing constrained PSP. The model was further enriched by incorporating realistic investment constraints – specifically the cardinality and floor & ceiling constraints.

Our decision to choose semivariance among other risk measures (such as Mean Absolute Deviation [91], Value at Risk [114] and many more [81, 145, 56]) was informed by the fact that, semivariance, being one of the popular downside-risk measures, according to Harlow [67] is attractive not only because it is consistent with investors' perception of risk, but also because asset allocation in downside-risk framework determines an investment opportunity set for downside-averse investors that is at least as efficient as that derived using the conventional E-V method. It should

also be noted that, right from the very beginning even Markowitz considered using an alternative measure of portfolio risk other than the variance he finally settled on – and this measure is none other than the semivariance. Markowitz [103] discussed it in detail and dedicated to semivariance an entire chapter (Chapter IX) wherein he stated: "Analyses based on *S* [Semivariance] tend to produce better portfolios than those based on *V* [Variance]". He went further in Markowitz [105] to claim that: "semivariance is the more plausible measure of risk". Later, he also claims in Markowitz *et al* [106] that, because "an investor worries about underperformance rather than overperformance, semideviation [square root of semivariance] is a more appropriate measure of investor's risk than variance".

The *E-S* portfolio selection framework replaces variance for semivariance (in the classical *E-V* formulation) as the measure of portfolio risk, thereby identifying those portfolios that seek to minimize/maximize semivariance/expected return for a given expected return/semivariance as efficient. The semivariance, according to Hogan and Warren [77] is devoted to loss reduction as opposed to the variance that considers "extreme gains as well as extreme losses as undesirable".

However, there is lots more to optimizing portfolios based on the *E-S* framework, in the sense that, unlike the well-known neat closed-form solutions obtained in the *E-V* PSP; the *E-S* problems are usually tackled by, what Estrada [47] referred to as, "obscure numerical algorithms". The main reason behind this is the fact that, as opposed to the exogenous covariance matrix incorporated as one of the main inputs in the *E-V* framework, the semicovariance matrix, one of the main inputs in the *E-S* framework is endogenous [See APPENDIX 1 for details].

With this in mind, our research seeks to estimate semivariance of portfolio returns in a similar way to that used in estimating variance of portfolio returns based on the expression proposed in Estrada [46, 47, 48, 49]. There are basically two main advantages in doing so; the first is that: the estimation of the semivariance of portfolio returns is made simple, convenient and as easy as estimating the variance – having in both cases equal number of inputs (means, variances/semivariances and covariances/semicovariances). The second is that, all this can be done with an expression popularly known by academics and practitioners alike without necessarily invoking the help of any *sophisticated* algorithm, and on top of that, the resultant portfolio semivariance is shown in Estrada [47], to be strongly positively correlated while at the same time being very close (in magnitude) to the actual value it tends to estimate.

Therefore, going by the Estrada's proposal (as detailed in equations 2.5(g) through 2.5(i)), our proposed model which is to be known as: The ***Mean-Semivariance (E–S) Portfolio Selection*** model can now be formulated as follows:

$$\textit{Minimize Portfolio Risk, } \Theta^2_{PB}$$

$$\textit{Subject to}$$

$$\sum_{i=1}^{n}\sum_{i=1}^{n} w_i w_j S_{ijB} = \Theta^2_{PB}$$

$$\sum_{i=1}^{n} w_i \, \overline{r_i} = R^*_P$$

$$\varepsilon_i \, \delta_i \leq w_i$$

$$\mu_i \, \delta_i \geq w_i$$

$$\sum_{i=1}^{n} \delta_i \leq k$$

$$\delta_i = binary \qquad \forall i = 1 \ldots n$$

Estrada [49] shows that, the value of the semivariance obtained in his semivariance formula (although a little bit different from the actual) can serve as a good approximation to the actual one as defined by Markowitz. With this in mind, we are now adopting the Estrada's formula for obtaining a semivariance rather than the Markowitz'.

3.0 Overview and Applications of Heuristics

3.1 Introduction on Heuristics/Metaheuristics

In everyday life, varying sectors in different aspect of human endeavour are faced with problems of growing complexity, arising in diverse spheres of life such as Operations Research, mechanical, electrical and electronic systems designs, image processing, signal processing and lots more. In all these sectors, the problems at hand can be formulated as an optimization problem, in which a single or several objective (cost) function(s) is/are desired to be optimized (either for minimization of cost or maximization of profit) subject to meeting or satisfying some conditions – which might be necessarily met (hard constraints) or met to some extent (soft constraints).

There are basically two known types of optimization problems, namely discrete and continuous problems [38, 72, 37, 39]. The most notable example under the discrete type is the well known Travelling Salesman Problem (TSP). An example of continuous optimization problems involves, according to Dreo *et al* [39] "the search for the values to be assigned to the parameters of a digital model of a process, so that this model reproduces the real behaviour observed, as accurately as possible". Many of the discrete and continuous optimization problems can be handled by some exact algorithms and solution to optimality is thereby guaranteed. Such algorithms include Simplex Algorithm (in Linear Programming Problems), Hungarian Method (for solving Assignment Problems), Johnson Method (for solving 2–machine sequencing problems), Branch & Bound, and Dynamic Programming.

However, there are some exceptional situations in which some other optimization problems are extremely difficult to solve by the conventional means; this is because,

the complexity of these problems grows with increase in the number of parameters, and the computing time therefore grows exponentially; these type of problems are commonly regarded as NP-hard. Similarly, some optimization problems of the continuous type may not have a known algorithm capable of finding the best possible solution (global optimum) within a reasonable period of time. For over two decades, there have been many unrelenting efforts and breakthroughs in various techniques that aim to provide solace to academics, practitioners and organizations in solving basically these two types of problems. In the field of discrete optimization, a reasonable number of heuristics were proposed, implemented and found to be effective in obtaining a solution close to the optimum, but many of them tend to be tailored towards a specific problem. Likewise, in the area of continuous optimization, most of the techniques developed tend to be ineffective, provided the objective (cost) function does not exhibit a particular structural pattern, such as convexity. As time passes by, computational power grows strongly coupled with constant dedication of academics and other industry specialists in their search for robust techniques capable of handling, not only discrete optimization problems but also their continuous counterparts; the emergence of metaheuristics (certain class of heuristics) signifies an important development in the world of optimization. Before we jump into exploring what metaheuristic techniques are all about, let us have a look at some basic definitions as follows:

Heuristic – coined originally from the Greek word *Heuriskein* which literally means to find or discover, is defined as "a technique which seeks good (i.e. near optimal) solutions at a reasonable computational cost without being able to guarantee either

feasibility or optimality, or even in many cases to state how close to optimality a particular feasible solution is"[123].

The term metaheuristic originated from the combination of two Greek words: a prefix – *meta* (meaning "beyond") and *heuristic* (meaning "to find" or "to discover"), and often regarded as a group of high-tech heuristic methods applied in solving problems with no known exact solution algorithms. It was however, believed to be developed, used and defined by Fred Glover as: "a master strategy that guides and modifies other heuristics to produce solutions beyond those that are normally generated in a quest for local optimality. The heuristics guided by such a meta-strategy may be high level procedures or may embody nothing more than a description of available moves for transforming one solution into another, together with an associated evaluation rule "[63].

Although, there still exist no common accepted definition of metaheuristics; it is in view of this, many researchers proposed several definitions. According to Osman and Laporte [118], metaheuristic can be defined as "an iterative generation process which guides a subordinate heuristic by combining intelligently different concepts for exploring and exploiting the search space, learning strategies are used to structure information in order to find efficiently near – optimal solutions."

The definition can also be viewed from another perspective as: A *rule of thumb*, based on domain knowledge from a particular application that gives guidance in the solution of a problem. Unlike algorithms, heuristics cannot have proven performance bounds owing to their open-ended dependence on specific application knowledge; an example

is "if the sky is cloudy, then carry an umbrella." Heuristics may thus be very valuable most of the time but their results or performance cannot be guaranteed [76]. In other words, metaheuristics refers to some classes of heuristic techniques, which are found to be applicable to virtually all types of discrete optimization problems, and can as well be adapted to the complex nature of continuous types. The term, may or may not be written as a hyphenated word, and basically refers to the collection of high-tech heuristic methods capable of offering practical solution to complex real life problems. These methods include Genetic Algorithms (GA), Simulated Annealing (SA) and Ant Colony Optimization (ACO), Tabu Search (TS), Particle Swarm Optimization (PSO), Iterated Local Search (ILS), Parallel Simulated Annealing (Par-SA), Threshold Accepting (TA) and others [15].

As mentioned above, there are basically several heuristic and metaheuristic techniques purposely developed to handle optimization problems especially, where the exact conventional methods are known to fail. Some of these techniques were tested and proven to be very good in finding optimal or near optimal solutions to real life problems. Metaheuristic algorithms, although proved to be strongly efficient in finding good and quite often near-optimal solutions, are unable to guarantee the optimality of the returned solution. Many among them, such as TS, ACO and some EAs are found to be very successful in solving real-world optimization problems partly due to their ability to conduct a guided local search using some intelligent criteria, while at the same time employing some mechanisms to escape being trapped in a local optimum. These successful criteria of escaping entrapment in local optima are mostly aimed at striking a balance between *intensification* and *diversification.*

The *intensification* mechanism also known as *exploitation* is aimed at exploiting the search experience by visiting and revisiting regions that appear to be promising in yielding high quality solutions and avoiding those that are already explored and found to be less attractive; whereas *diversification*, often referred to as *exploration*, has to do with exploring new search space regions that were not visited before with the hope of finding better solutions than the ones previously found in other regions of the search space.

Metaheuristic algorithms may be classified according to different features in their mode of search operations. Some metaheuristics might be *guided/unguided, single-agent based/multi-agent based, deterministic/stochastic, nature-inspired/nonnature-inspired, iterative/greedy, trajectory/non-trajectory.*

Guided search methods are intelligence-tailored and memory-conscious algorithms that incorporate some additional strategies and hints about where the search should focus in the search space. For example, TS is a *guided* local search algorithm in the sense that, it stores a database of recently visited solutions in a Tabu List, thereby avoiding cycling and easy entrapment in a local optimum. Similarly, the same can be said about ACO in which traces of pheromone represent an adaptive memory of previously visited solutions. Contrarily, the *unguided* search algorithms, such as SA are, in fact, memoryless in the sense that no information extracted dynamically is used during the search, thereby only relying on the search processes' behaviour without any additional help or hint.

Nature-inspired methods are algorithms that were inspired as a result of analogies with some aspects of natural processes. For instance, GA, EA and artificial immune systems (AIS) come from Biology, while ACO and PSO are from Ethology. *Nonnature-inspired* are those that result from some processes in human endeavour. A typical example of a nonnature-inspired strategy is SA that was derived from an analogy with physical processes in Physics.

Some algorithms are *deterministic* (e.g TS) while others are *stochastic* (e.g SA). The former set of algorithms solves an optimization problem through taking some deterministic decisions, and this enables them to arrive at the same final solution when using the same initial solution in different runs. The later set apply some random rules during the search which enables them to explore the solution space in a stochastic or non-deterministic manner with the primary aim of finding a better (global) solution than the current (local) one. Thus, in stochastic metaheuristics, varying final solutions may be obtained in different experimental runs, even if the search started (in all cases) from the same initial solution.

In *iterative* algorithms, the search begins with a complete solution or set of solutions which are perturbed and transformed at each iteration using some set of search operators with the hope of obtaining better solution(s). *Constructive* algorithms, on the other hand, begin their search from an empty solution upon which at each step a decision variable is assigned until a complete solution is arrived at. It is noteworthy that the majority of metaheuristics are *iterative* algorithms. *Constructive* algorithms tend to be myopic in their way of solution construction, as their look-ahead ability is short-sighted and the consequences of their decisions can only be felt in the future.

The *single-agent* methods (such as SA, TA and TS) mostly perturb and manipulate a single solution at any point in time during the search trajectory, while the *multi-agent* methods (such as GA, PSO and ACO) allow for the participation, evolution and collective effort of several solutions in the search processes thereby contributing to the success of the entire search independently and in parallel. These two sets of algorithms are believed to have complementary features, in the sense that, the *single-agent* based metaheuristics are exploitation oriented; this is because they are capable of intensifying search for a better solution in the local region. On the other hand, *multi-agent* (population) based methods are exploration oriented as they allow for thorough diversification in the entire search space.

The *trajectory* methods find the next solution by partial or exhaustive search of the immediate neighbourhood of the current solution. The next (candidate) solution can be obtained by slightly perturbing the configuration of the current solution. Typical examples of *trajectory* methods are SA, TS and TA. However, for *non-trajectory* methods, it is possible for the next solution to be far from the current solution as there are possible *jumps* in how they are generated, for instance due to the influence of say, genetic operators in GA.

To obtain some more details on metaheuristics, an interested reader should consult Blum and Roli [11], Osman and Laporte [118] and Dreo *et al* [39].

3.2 Applications of Heuristics/Metaheuristics

Metaheuristic techniques are found to be very applicable and of immense importance in solving combinatorial (both discrete and continuous) optimization problems and as

such their importance and centrality across much research in the optimization community (both in academia and industry) can never be overemphasized. In many researches, they were found to be very promising in finding good and most of the time near-optimal solutions within a reasonable period of time [15]. Heuristics and metaheuristics have been used for a very wide variety of real problems; small samples of these are discussed briefly below.

Henderson *et al* [73] used SA and proposed a model that would be used to solve the shortest route cut and fill problem (SRCFP). The model was used to find an optimal shortest route to be followed by an *earthmoving* vehicle on a construction site characterized by several *abnormal* terrains. The algorithm developed is aimed at minimizing the total distance covered by the vehicle in levelling the site to get what is known as *final grade* site suitable for construction; and this, will consequently lead "to saving costs of fuel consumption, equipment maintenance and time."

Another important and one of the recent studies showing the capability and robustness of heuristic techniques in handling and solving complex combinatorial optimization problems can be found in the research conducted by Xiang *et al* [150]. They considered an algorithm to solve a large scale static "dial-a-ride" problem using the intensification and diversification strategies well known in TS metaheuristic technique. Xiang *et al* [151] is an extension or rather a dynamic approach to the algorithm in Xiang *et al* [150], and this approach is believed to be capable of generating high quality schedules amid challenges in handling various stochastic events.

Erera *et al* [44] solved a *Driver Scheduling and Load Dispatching Problem* (DSLDP), using a heuristic technique combining greedy search with enumeration to obtain a cost effective solution of scheduling problem for less-than-truckload (LTL) carriers having up to 15,000 – 20,000 dispatchable loads by few thousand drivers in a reasonable computing time. The DSLDP was found "to be applicable not only to LTL carriers, but to small package express carriers". Briant *et al* [14] used another variant of SA to solve a challenge organized by French Society of Operations Research and Decision Analysis (ROADEF) tagged as ROADEF'05 challenge. The problem topic for the said challenge was Car Sequencing problem. The new variant used is known as *multi-criteria dynamic simulated annealing* partly because it computes the various probabilities of acceptance dynamically. In the classical car sequencing problem, the violations of the total number of spacing requirements between some vehicles characterized with some options has to be minimized. However in this challenge, the problem had two level of spacing requirements which was optimized by the SA variant one by one in their order of importance.

Another research attesting to the wider applicability, flexibility and robustness of metaheuristic techniques in solving diverse optimization problems was conducted by Hu [80]. The research focused its searchlight on TS's *reliability* and *efficiency* in solving to optimality some engineering design problems. The technique was compared to, and was found to outperform two other metaheuristic techniques, namely random search and genetic algorithm, for the selected continuous variables test problems.

Another research was conducted by Siarry and Berthiau [132] to primarily investigate the capability of TS metaheuristic in optimizing a set of classical continuous multi-

minima functions with known global optima. The technique used was aimed at proposing an "adaptation" of basic TS algorithm to the optimization of continuous functions and at the same time investigate the influence of the algorithm's parameters upon convergence to the desired optimum. The procedural structure of the research employs the notion of *balls* in defining the neighbourhood of a solution and the v neighbours of the current solution are randomly selected inside a *ball* [**B**(c, r) | c is the centre of the ball and r the radius]. The entire solution space is partitioned into a set of concentric *balls* with radii $r_0, r_1, \ldots, r_v$. The v neighbours are generated by randomly picking a single solution from each of the v concentric *balls* before being checked for tabu membership; and if any of the solutions generated was found to be in the tabu region it is discarded and another solution is then selected from the same considered *ball*. To make sure that the concept of diversification is implemented and the algorithm can escape being entrapped in a local minimum, the immediate neighbourhood [**B**(s, r_0)] of the current solution is excluded in generating the candidate neighbours. As in the conventional TS procedure, the *best* of the v non-tabu neighbours of the current solution becomes the new current solution even if the objective function is worsened.

A research conducted by Cvijovic and Klinowski [27] extended the conventional TS as proposed by Glover to tackle some continuous-valued functions. They studied the potential of their modified TS algorithm in solving multivariate continuous functions characterized by many local minima. In their quest to come up with a robust, efficient and effective algorithm; they introduced and implemented a neighbourhood structure tagged as *conditional neighbourhood.* The entire search space is compartmentalized into a number of disjoint cells by dividing the coordinate intervals along the x's axes

into p parts. Throughout the study, two kinds of tabu moves were implemented as follows:

(i) A particular move in the neighbourhood of the current solution is regarded as being tabu (not allowed), if the newly generated solution lies within the tabu region of the search space consisting of cells visited during the last L iterations; this type of move is managed by tabu list.

(ii) A move is tabu also, if it results in worsening the objective function f more than some specified (threshold) value. This move is managed by "keeping the track of the worst value of the objective function f throughout the computation and maintaining the '*elite list*' of addresses of the most promising cells".

The concept of aspiration criterion was also introduced by tracking the best ever found value of the objective function, in which the tabu status of a newly generated candidate solution is overridden if the aspiration condition is satisfied [that is, f(new solution) <= Aspiration Function]. It was reported that, the algorithm designed was a successful one, since after the average of 100 independent runs were taken, the reliability was found to be excellent, due to the fact that, in at least 90% of the runs conducted, the final results obtained lie within 2 to 3% of the global optimum. It is also easily observable in the results presented that, among all the heuristic methods with which the tabu search is compared, it had the least number of function evaluations before arriving at a global minimum solution over the six multivariate continuous functions tested.

The procedure implemented in this research is regarded to be "generally applicable, easy to implement, derivative-free, and conceptually simple".

3.3 Heuristics in Multi-Objective Optimization problems

There is, undoubtedly, an increase in interest for scientific research involving multi-objective optimization; and this is not unconnected with the fact that, in many real-life problems (such as engineering, construction design, finance, etc), there are quite often numerous objectives that need to be achieved. In such multiple objective problems there exists no singular best solution, but rather a collection of solutions that are better than others when all the objectives are taken into consideration. Thus, no universal optimal solution in such a context exists, in the sense that whenever an attempt is made to improve one of these objectives, there will be a consequent degradation of one or more other objective(s). The explanation for the multiplicity of these solutions lies in the conflicting nature of those objectives.

Apart from the fact that, modelling a solution for a single-objective optimization problem can prove to be a difficult task; there is also the possibility that, the goal of modelling such a single-objective problem can be spoiled by a bias during the modelling stage. On the other hand, multi-objective optimization techniques offer some degree of freedom which cannot be found in modelling single-objective optimization problems. However, this flexibility comes with a cost, especially on the method used to solve the problem when it is finally modelled. The search often does not give a unique solution, but rather a set of solutions. The main idea behind multi-objective optimization lies solely in searching for a set of agreements among the various problems' objectives; and the final decision about which solution or set of solutions is to be chosen lies entirely with the final user of the results generated.

To illustrate the concept briefly, suppose one wants to buy a property (House), H; some of the things one might consider giving priorities to, include: price, P; state/goodness of the property, S and location, L. Therefore, a property H having price P, in a good condition S and located at L is *better* than another property H^1 with price $P^1 > P$, whose condition $S^1 < S$, but located in the same area L. On the other hand, the same property H cannot be compared with another property H^2 whose price $P^2 > P$ in an extremely good condition $S^2 > S$, located in the same neighbourhood, L as H.

So more formally, a brief mathematical description of multi-objective optimization can be defined as follows:

Let $\left[X = (x_1, x_2, ..., x_n)\right]$ be some decision variables of a given problem; while $\left[F(X) = (f_1(X), f_2(X), ..., f_m(X))\right]$ be some set of objective functions to be optimized. A multi-objective optimization can be defined as:

$min\left[f_1(X), f_2(X), \ ..., \ f_m(X)\right]$	(1)
subject to	
$c_1(X) \le b_1$	(2)
$c_2(X) \le b_2$	(3)
$\ddots \ddots$	
$c_r(X) \le b_r$	(r)

A given solution $\left[X = (x_1, x_2, ..., x_n)\right]$ is said to be *nondominated*, provided no other solution can be found to improve $\left[X = (x_1, x_2, ..., x_n)\right]$ for a given objective $f_i(X)$ without necessarily worsening at least one of the other objectives.

Now given a multi-objective optimization problem; a given solution, X' is said to *dominate* another feasible solution X'', for that *if* $f(X_i') \leq f(X_i'')$ $\forall i$; *and at least there exist* $j \in 1, 2, ..., n | f(X_j') < f(X_j'')$. The set of nondominated solutions constitutes what is normally regarded as *pareto-optimal, pareto-border* or *pareto-front* solutions. Having traced out the pareto-border, the decision maker would then be faced with some difficulties of selecting a solution from such set of solutions; the solution being the one that reflects the decision maker's tradeoffs or preferences in relation of the various objective functions.

Another definition given by Alaya *et al* [3], defined the formal representation of a multi-objective optimization problem by a quadruplet (*X, D, C, F*) in which *X* signifies a vector of *n* decision variables $\left[X = (x_1, x_2, ..., x_n)\right]$; *D* signifies a vector of the decision variables' domains $\left[D = (d_1, d_2, ..., d_n)\right]$; while *C* is the set of constraints on *X* and *F* is the vector of $m \geq 2$ objective functions, $\left[F(X) = (f_1(X), f_2(X), ..., f_m(X))\right]$; without loss of generality, these objective functions are assumed to be minimized (for those to be maximized may be multiplied by -1).

The space of candidate solutions, noted *E(X, D, C)* is the set of vectors $v \in D$ satisfying all the constraints of C. We define a partial order relation on this set as follows: a solution $v \in E(X, D, C)$ dominates a solution $v' \in E(X, D, C)$, noted $v \prec v'$, if and only if v is at least as good as v' for each of the *m* criteria to optimize, and strictly better than v' for at least one of these criteria; that is, if and only if $\forall i \in \{1, ...,$

$m\}$, $f_i(v) \leq f_i(v')$ and $\exists i \in \{1, ..., m\}$, $f_i(v) \leq f_i(v')$. The goal of multi-objective optimization problems is to find the Pareto optimal set of all non-dominated solutions, i.e., $\{v \in E(X, D, C) \mid \forall v' \in E(X, D, C), \neg(v \prec v')\}$.

The search for an efficient algorithm to tackle multi-objective optimization problems still continues in the scientific research community, partly because we are yet to have a better grasp of the existing relationship between algorithms' design and their performance on some specific problems. Recently, the usage and successful implementation of metaheuristics and evolutionary techniques in solving multi-objective problems have become very popular, due to: (i) their ability to provide good multiple solutions in a single run, (ii) their convergence speed and degree of accuracy in estimating the pareto-optimal solutions, (iii) ability to easily handle both continuous and discrete optimization problems, (iv) their ability of being less susceptible to the (dis)continuity of the pareto-border. Lin and Kwok [97] applied TS and SA techniques to solve a location-routing problem (LRP) in which multi-objective decisions on location of depot, vehicle routing and assigning routes to vehicles were considered concurrently. Alaya *et al* [3] proposed an ACO algorithm entitled *m*-ACO designed for solving multi-objective optimization problems. The 4 variants of the algorithm were tested on multi-objective knapsack problem (MOKP) against several EAs proposed in the literature for solving the MOKP. The results presented showed that one of the variants outperformed all the EAs it was compared with. Mohamed *et al* [112] proposes a Bi-criteria Genetic Algorithm for solving Bicriteria Shortest Path Problem (BSP) in which two conflicting objectives: minimizing the transportation cost and the total travel time were considered.

Abido [2] proposed a multi-objective PSO (MOPSO) technique for solving environmental/economic dispatch (EED) problem with competing objectives on minimizing cost and emission. The algorithm was so successful that it was capable of "generating a set of well-distributed Pareto-optimal solutions in one single run". Armananzas and Lozano [4] on the other hand, tackled PSP from a multi-objective point of view using three well known metaheuristic techniques, namely the greedy search, SA and ACO. They made use of the capital market indices data made publicly available at the OR Library [117]. Their results indicated that ACO and SA performed better than the greedy search method in all the five instances considered. Ghoseiri and Nadjari [58] presented an algorithm based on multiobjective ACO to tackle a bi-objective shortest path problem.

Baykasoglu [8] proposed a multi-objective TS (MOTS) that can be applied to several goal programming problems. The proposed algorithm (MOTS) was found to be efficient and effective in solving four test studies out of which two are difficult engineering design problems collected from the literature. Feng *et al* [54] proposed a multi-objective particle swarm optimization based on crowding distance sorting (CDMOPSO). In order to verify how well the newly proposed method performed in relation to existing methods in the literature; it was tested with six unconstrained and three constrained two-objective test problems and the results compared against those obtained by two well known methods, namely Non-dominated Sorting Genetic Algorithm II (NSGA-II) [33] and Strength Pareto Evolutionary Algorithm 2 (SPEA2) [158]. In the unconstrained cases, the CDMOPSO was found to have better convergence ability as well as diversity maintenance capability in relation to the other two methods it was compared with. Moreover, it was reported that, the method *gained*

a very good effect when it was applied to another multi-objective optimization problem of a large scale injection machine.

Suman et al [136] proposed a SA based multi-objective optimization algorithm entitled Orthogonal Simulated Annealing (OSA). The new proposed algorithm was tested on some multiobjective problems with different degree of complexity; against a popular multiobjective evolutionary algorithm (SPEA2 [158]) and another one called classical simulated annealing based multiobjective algorithm (CMOSA [137]). The authors reported that, the new method was such a success that, it was able to outperform the other two methods in some of the tested problems in relation to performance and CPU time. Moreover, apart from its apparent ability of obtaining well diversified set of solutions and capturing the Pareto front better than the CMOSA; it was, particularly, found to outperform CMOSA in around 70% of the times.

3.4 Metaheuristics in Portfolio Selection

The unconstrained Markowitz Mean – Variance model can be regarded as a simple quadratic optimization problem for which there exist computational algorithms that effectively handle the problem and computing an optimal solution for any large data set is not difficult. As already mentioned previously, Variance as a measure of portfolio risk has been criticized by several researchers and this led to introducing some alternatives. However, introducing alternative risk measures alone is not sufficient to fix the flaws inherent in the original Markowitz model, as there are other issues to do with investment constraints; in which case any attempt to incorporate these realistic practical constraints into the original model has some accompanying consequences, such as the transformation of the problem from a mere convex

nonlinear optimization problem into a computationally-costly non-convex NP-hard combinatorial optimization problem which cannot be solved by any known algorithm in a polynomial time. With these consequences, the problem can only be handled in a practical time domain by approximate (heuristics/metaheuristics) algorithms.

Some researches proposed an alternative to variance while at the same time incorporating realistic constraints into their portfolio selection model. For instance, Speranza [134] linearized the objective by introducing Mean Deviation below Average and using techniques involving Branch & Bound to solve a constrained formulation of the problem after incorporating additional constraints dealing with transaction cost, transaction units, cardinality constraints and integer variables. Hamza and Janssen [68] used separable programming techniques to solve the constrained version of the problem while adopting semivariance as the objective and maintaining all the constraints introduced in Speranza [134] except for the cardinality restriction. Bienstock [10] approached a cardinality constrained *PSP* by introducing some valid inequalities (cuts) and tested a self-developed branch-and-cut algorithm based on disjunctive cuts. The algorithm's computational results involving up to 3897 assets were presented. Lee and Mitchell [94] also solved a cardinality constrained *PSP* formulation. Their method, based on an interior point nonlinear solver, was used to solve problems involving up to 150 assets.

Dueck and Winker [40] solved an instance of PSP with semivariance as a risk measure using a local search method called threshold accepting (TA) which, in principle, is similar to SA. Gilli *et al* [59] used TA algorithm to minimize value-at-risk and expected shortfall. Chang *et al* [20] applied 3 prominent metaheuristic techniques

(GA, TS and SA) to solve the constrained version of the original model by introducing 2 additional realistic constraints – the cardinality and floor & ceiling constraints. They tested their algorithms with 5 datasets, each with varying number of assets reaching up to a maximum of 225 assets. They reported that, best results are obtained by pooling the optimal solutions from all the three algorithms. Crama and Schyns [25] used SA algorithm to solve the model tackled by Chang *et al* [20] but with additional turnover (purchase & sales) and trading constraints.

Jobst *et al* [84] solved a PSP by combining a branch-and-bound algorithm with some heuristic methods (*integer-restart* and *re-optimization heuristic*) to specifically make a comparison with the results presented in Chang *et al* [20]. The first heuristic (*integer-restart*) plotted the constrained efficient frontier, beginning from the highest return through to the lowest return. The procedure implemented is such that, the current stage uses as initial solution, the result obtained in the preceding stage. This heuristic is named as *warm restart* heuristic. The other heuristic inspired by the idea similar to the one implemented in Speranza [134] initially solves a continuous relaxation excluding any constraint and then uses as inputs, the K assets with highest weights for a problem where constraints are imposed. The two heuristics were embedded in a branch-and-bound algorithm and were reported to have performed better than the metaheuristics implemented in Chang *et al*. However, the *re-optimization heuristic* normally fails to plot the frontier when fewer than k assets are produced by the continuous relaxation.

Another study by Fernandez and Gomez [55] compared the performance of a neural network approach to the three metaheuristic techniques used in Chang *et al* [20]. They use a neural network having a single layer of fully connected neurons (Hopfield

network) to plot an approximate constrained efficient frontier (ACEF) after the imposition of cardinality and bound constraints. The results obtained show no significant difference between their neural network approach and the results obtained from the metaheuristics presented in Chang et al. In order to obtain an improved ACEF, they adopted the idea used by Chang *et al* to wipe out the dominated portfolios after pooling the portfolios obtained from the four approaches. By so doing, the quality of solutions obtained significantly improved, hence making their neural network approach to solving a PSP a success. Although the neural network donated the largest number of portfolios in the new frontier, it is clear that a stand-alone neural network is unsuitable for solving the problem over the entire efficient frontier.

Kendall and Su [86], used PSO techniques to solve a PSP involving some risky and risk-free assets, in which the main goal was to maximize what they referred to as the *reward-to-variability ratio* on various constrained and unconstrained portfolio investment problems. The algorithm was found to outperform the classical Excel Solver in most of the experimental runs; however, it exhibited a "high computational efficiency in constructing optimal risky portfolios of less than fifteen assets" only. Schaerf [129] proposed new algorithms based upon TS to solve the constrained PSP tackled in Chang *et al* [20] by combining and testing several neighbourhood relations. Streichert *et al* [135] investigated the capability of EAs to solve the constrained PSP incorporating Cardinality Constraints, Buy-in Thresholds and Roundlots constraints.

Chen *et al* [21] extended the classical *PSP* by incorporating transaction costs and floor & ceiling constraints; experimental results reported involved only eight different stocks data downloaded from a Chinese Financial market. Cura [26] applied *PSO* to a

constrained *PSP* incorporating cardinality and floor & ceiling constraints. The results reported by the *PSO*-based heuristic method were compared to the earlier study conducted by Chang *et al* [20]. Although, none of the four compared heuristics seem to significantly outperform the rest, the research concluded by giving credit to the *PSO*-based heuristic in the sense that, it was able to give better solutions than the other methods "*when dealing with problem instances that demand portfolios with a low risk of investment*".

3.5 Overview on some chosen Metaheuristics

We have chosen, designed and implemented 6 different metaheuristic algorithms, among which 2 of them (SA and TS) are local search techniques, 2 (GA and PSO) are EAs, 1 (Parallel SA) a parallel implementation of SA and 1 (SWAN) a hybrid of SA & PSO. All the aforementioned algorithms were designed and coded in C++ programming language (which we learned specifically to conduct this research) and executed on Dell's Desktop Computer with an *x86 Family 6 Model Stepping 2 GenuineIntel (~2126MHz)* processor under *Microsoft Windows XP Professional* Operating System.

3.5.1 Simulated Annealing (SA)

SA is one of the oldest, well known and widely applied local search metaheuristic techniques used in solving combinatorial optimization problems [11]. Although, it was originally developed in statistical mechanics based on a Monte Carlo model by Metropolis *et al* [107] to simulate the processes involved in heating and cooling of a solid material; it was however, Kirkpatrick *et al* [90] and Cerny [19] independently in the early eighties who noted similarities between the physical process of annealing and

some combinatorial optimization problems. They observed an interesting correspondence between the physical state of metallic materials and the solution space of an optimization problem. They further observed that the objective function in optimization problem corresponds to the free energy of the material. Similarly, an optimal solution corresponds with a defect-free crystal, whereas a crystal with defects corresponds to a local optimal solution. However, not all the analogies observed are based on one-to-one correspondence. For instance, the annealing process involves the usage of a physical variable – a *temperature* which when monitored under proper control plays an important role in obtaining a perfect crystal. But when using SA in solving an optimization problem, the *temperature* just serves as a control parameter that has to be properly determined and continuously adjusted, which after a long run plays a vital role in obtaining a very good solution. These sets of observations (and many more) led to some series of publications that brought SA to the limelight in the combinatorial optimization community.

SA derived its name from an analogy in the process of physical process of solids, whereby a crystalline solid is heated to a melting point (i.e. to a very high temperature, in which the particles in the solid moves freely and haphazardly without any definite direction), and later allowed to cool carefully at a very slow rate up to the point it (freezes) reaches its most regular crystal lattice configuration (i.e. until the particles arrange themselves in the ground state of the solid), which consequently leads to a resultant solid free of any crystal defects. In the processes involved in such cooling procedure, it is assumed that the thermal (or quasi-) equilibrium conditions are maintained, and the processes end when the material reaches its minimum energy state, which in principle, corresponds with a perfect crystal. The cooling procedure has

to be slow and steady in order to obtain a defect-free crystal (i.e. minimum energy solids). Transition mechanism between different states and the cooling schedule are what constitute the main and most important features of an SA algorithm.

SA, also known as *statistical cooling*, *Monte Carlo annealing*, *probabilistic hill-climbing*, *stochastic relaxation* and *probabilistic exchange algorithm* [109], is often regarded as one of the most flexible methods for tackling difficult combinatorial optimization problems [113]. A vital feature of SA is its usage of the so-called *hill-climbing* moves (which worsen the objective function value) purposely made in search of global optimum (or specifically aimed at escaping entrapment in a local optimum) especially in a solution space characterized by several local optima. SA is believed to be one of the early algorithms that had a clear and laid down path to escape being trapped in a local optimum.

The algorithm has lots of advantages that informed our decision to choose it as one of the metaheuristic techniques to be implemented in this research; some of which are:

(i) Its ability to (statistically) guarantee finding optimal solutions.

(ii) Although, time consuming, it is relatively easier to code than some other methods.

(iii) As Eglese [43] argues, SA provides good (and not necessarily optimal) solutions.

(iv) Its wider applicability to large optimization problems irrespective of the differentiability, continuity and convexity conditions that are normally required in conventional optimization methods.

(v) It does not assume any particular property (such as linearity or convexity) of the problem at hand.

There are basically some theoretical fundamental issues to consider in the implementation of SA algorithm, and these include:

3.5.1.1 Metropolis Algorithm

The Metropolis algorithm is the original and most vital idea behind the SA algorithm, which (through Monte Carlo simulation) models the microscopic behaviour of some set of large number of particles, as in solids [113]. In the field of thermodynamics, any material has individual particles with varying energy levels according to a certain statistical distribution; the minimum energy level (often regarded as *fundamental level*) occurs normally at temperature 0K, and at this level all particles are believed to be in a stand still position. However, the particles possess different energy levels as the temperature increases above the fundamental level; thereby leading to a decrease in the number of particles that roam about at higher energy levels (this implies the maximum number of particles occurs at the fundamental level). It should be noted that, the statistical distribution of these particles in the various energy levels varies with the temperature and the number of particles is a decreasing function of the energy level.

The (Metropolis) algorithm, given a solid in state S_i with energy E_i, generates a sequence of states S_j through a transition mechanism involving minor changes to the original state achievable by moving one of the solid's particles according to the Monte Carlo method. Suppose the energy of the resultant state also found based on some

probability be denoted by E_j; now, if E_j is *less than or equal* to E_i, the newly generated state S_j is accepted, otherwise it is only accepted with a probability given by:

$$\exp\left(\frac{E_i - E_j}{k_B T}\right) \qquad 3.5.1.1$$

The T stands for the temperature of the solid, while k_B is known as the Boltzmann constant. Another name for this kind of acceptance rule is known as *Metropolis criterion* and the entire algorithm described above is the *Metropolis* algorithm.

It should be noted at this point that, the thermodynamic equilibrium for the current temperature can only be achieved (before moving to the next level) when the rate at which the temperature is changed is carefully chosen, and this often requires a sizeable number of states transitions of the Metropolis Algorithm.

3.5.1.2 Cooling Schedule

The efficiency and effectiveness of SA algorithm (regarding the quality of the final solution and the number of iterations) in solving certain optimization problems largely depend on the choice of some control parameters, collectively known as the cooling schedule. Cooling schedule, as a control strategy used in guiding the algorithm from the beginning until convergence to an optimal or nearly optimal solution, is characterized by four different parameters as follows:

(i) Initial value of the temperature, T_o

(ii) Determination of a cooling rate, λ.

(iii) A finite length of each homogeneous Markov chain.

(iv) Final value of the temperature (stopping criterion).

Numerous pieces of research have been conducted, (see for instance Aarts and Korst [1]) and some might still be ongoing to come up with an *adequate* and *acceptable* cooling schedule. It should be noted at this point that, the performance of a cooling schedule is entirely and highly dependent upon the problem at hand. There are two main classes of cooling schedules, categorized into "static" and "dynamic" [72, 1].

3.5.1.2(a) Static cooling schedules:

Implementing an SA algorithm while employing *static cooling schedules* means that, the values taken by the set of parameters (mentioned above) must be wholly specified at the initial stage of the algorithm, remain fixed and cannot be changed during the execution of the algorithm. This is the pioneering cooling schedule – often referred to as *geometric cooling schedule,* used by Kirkpatrick *et al* [90], and still applied in many optimization problems.

Although, there are no generally acceptable guidelines or rules for choosing the values of the cooling schedule parameters, even if the classical *geometric* cooling schedules are to be used; however, the following tactical decisions have to be made in order to keep our algorithm effective and operational throughout the execution of the entire search process.

(i) Initial value of the temperature: The *temperature* parameter is a non-increasing function of time. The parameter's initial value should be chosen in such a way that, it is *sufficiently* large enough to allow for the proper exploration of the

solution/search space by ensuring acceptance of worse moves with a certain high probability at the beginning of the search process [43]. There are, however, several suggestions in the literature of how the initial temperature value should be selected. For further details, we refer the interested reader to Henderson *et al* [72], Dowsland [37], Monticelli *et al* [113], and Aarts and Korst [1].

(ii) Determination of a cooling rate: It is believed that, number of approaches exists in the literature on executing a *temperature* reduction in SA. Typical example of a static cooling function is given by:

$$T_{i+1} = \lambda T_i, \qquad i = 0, 1, \ldots$$

Where λ (cooling rate) is a positive fixed value smaller than, but close to, unity and whose typical values lie in the interval: $80\% \leq \lambda \leq 99\%$. For further details, see Eglese [43], Dowsland [37, 38] and Aarts and Korst [1].

(iii) Length of Markov chain: Often denoted by N_k, this simply means, the number of neighbourhood moves to be conducted under each temperature level. This parameter is more closely related to the cooling rate, λ than any other. Some proposals for determining this length, involves fixing the value, making it vary, setting it proportional to the problem dimension or proportional to the size of the neighbourhood defined [43, 1].

(iv) The Final value of the temperature: This basically serves as a stopping criterion for most implementations of SA algorithm. It is the temperature value whom upon assumption forces the cessation of program execution, and consequently

termination of the run of the processes. Most of the time this value is set at some small fixed value related to smallest possible difference in cost between two neighbouring solutions.

3.5.1.2(b) Dynamic cooling schedules:

Dynamic cooling schedule in relation to the static one is more complex in any implementation of an SA algorithm. It involves setting the initial and final value of the temperature parameter, the cooling rate and the length of the Markov chain within each temperature in a more dynamic way. There are, however several extensions to the *static* cooling schedules, that give rise to *dynamic* (variable) cooling schedules.

For instance, Reeves and Beasley [123] suggested that, the *initial temperature value* should be obtained by:

$$T_0 = \frac{\alpha}{-\ln \beta} f(x_0)$$

In which it is assumed that β% of the uphill moves, which are α% worse than the initial solution $f(x_0)$, are accepted at the initial temperature level T_0.

Monticelli *et al* [113] however, suggested three alternative ways for implementing *temperature reduction* in SA, among which (under dynamic cooling rate) one can use:

$$T_{i+1} = \frac{T_i}{\exp\left(\frac{\lambda T_i}{\sigma(T_i)}\right)}, \qquad i = 0, 1, \ldots$$

Where $\lambda \leq 1.0$ and $\sigma(T_i)$ is the standard deviation of the costs of the configurations generated at the previous temperature level T_i.

Furthermore, Monticelli *et al* [113] mentioned that, under dynamic cooling schedule, the length of Markov chain can be set according to:

$$N_{k+1} = \rho N_k, \qquad k = 0, 1, \ldots$$

Where ρ is a user-supplied parameter typically taking a value less than or greater than one. It should be made categorically clear that, whichever type of cooling schedule one decides to adopt in implementing a SA algorithm; according to Eglese [43], it is extremely important for the algorithm to spend less time at extreme (higher and lower) values of the temperature. This is because, at higher values, most of the worst solutions generated are accepted and staying there for long results in wasting precious run time. While at lower temperatures most of the neighbourhood moves are rejected, and it is worthwhile checking whether a local optimum has been attained.

3.5.1.3 Algorithmic implementation of SA

The procedure for executing SA algorithm can be described as follows:

Let Θ be a set of all possible solutions (solution space) and let $f : \Theta \rightarrow R$, be an objective function defined on Θ, the goal is to determine a global minimum $s^* (s^* \in \Theta \ \ such \ \ that \ \ f(s_i) \geq f(s^*), \forall s_i \in \Theta)$. For the global minimum, s^* to exist, the objective function, f must be bounded. By defining $N(s)$ as a neighbourhood function for $S_i \in \Theta$, it means for every solution $S_i \in \Theta$ there are

neighbouring solutions $s_j \in N(s)$ reachable in a single iteration of a local search algorithm.

The SA algorithm always starts with an initial solution, $s_i \in \Theta$ after setting the *temperature* parameter T to an initial value T_0 and an initial number of iteration, N_0. A neighbouring solution, $s_j \in N(s)$ is then generated (either stochastically or using some pre-specified rule). If we denote the difference between the initial and the newly generated objective function by δ [*i.e.* $\delta = f(s_j) - f(s_i)$], the newly generated solution, s_j is accepted as the next/new current solution depending on the value of T and δ. After computing δ, if the objective value got better $(i.e.\ \delta \leq 0)$, then the current solution s_i is substituted with the newly generated solution s_j. Otherwise, a random number, $rand \in U(0,1)$ is generated, and s_j can have another chance of replacing s_i as the new current solution, if and only if *rand* is less than some threshold value: $\exp\left(\frac{-\delta}{T}\right)$. This uphill move ability enables SA to escape entrapment in a local optimum.

The above described procedure continues in a repetitive fashion until a global optimum or a desired solution is obtained. The following pseudocode on figure 2 on the next page summarizes the detailed processes involved in an SA algorithm:

```
Simulated Annealing(inputs: s0, T0, λ, N; output: s*)
Begin
  Initial solution s0                    % choose an initial solution, s0 ∈ Θ
  Initial temperature T0                 % choose an initial temperature T = T0
  Choose a cooling rate λ                % choose T reduction function
  Choose N                               % define the length of Markov chain per T
  s = s0                                 % set the initial solution as the current
  Repeat Procedure
    k = 0                                % Initialize iteration counter to null
    For k = 1 through N                  % Loop until iteration counter equals N
      Generate(sj ∈ N(s))                % choose a solution sj from the neighbourhood N(s) of s
      compute δ = f(sj) − f(s)           % δ is the difference in objective btw the current & new solution
      If (δ ≤ 0) then                    % given that the objective function got better
        s = sj                           % set the new solution sj as the current s
      Else                               % if no any improvement
        Generate[r ∈ U(0,1)]             % generate a uniformly distributed random number btw 0 & 1
        If r < exp(−δ / T) then          % if this condition is satisfied
          s = sj                         % set the new solution sj as the current s
        End(If)
      End(If − Else)
      k = k + 1                          % increment iteration counter
    End(For)
    T = λ(T)                             % update the temperature T
  Until stopping criterion
  return s*                              % output the best / global solution found, s*
End
```

Figure 2: Typical pseudocode of an SA Algorithm

From the SA pseudocode it can be inferred that SA is a technique or method used in solving an optimization problem by iteratively perturbing the current solution in a stochastic manner. The method always accepts a local ascent/descent (depending whether it is a maximization/minimization problem). However, in order to escape entrapment in a local solution or to explore some other unexplored areas of the search

space, the method occasionally accepts a deteriorating solution as the next current solution with a given probability that decreases as the process continues. If the reduction rate – known as a *cooling schedule* – at which the "temperature" decreases is sufficiently low, there is a very strong likelihood that the algorithm would eventually arrive at a very good solution; however this achievement is most of the times at the expense of a longer run time. This type of local search escapes getting trapped in a local optimum by jumping out of it in the early part (i.e. at higher temperatures) of the search. As the number of iterations increases, the temperature and as well, the probability of accepting a worse solution approaches zero, and consequently the algorithm targets the bottom of a local optimum.

For more details on SA algorithms, we refer any interested reader to Aarts and Korst [1], Reeves and Beasley [123], Dreo *et al* [39], and/or Dowsland [37, 38].

3.5.2 Parallel Simulated Annealing (Parallel SA)

The parallelization of SA has been studied and found to be promising in several researches conducted across diverse research areas including global optimization as in Onbasoglu & Ozdamar [116], chromosome reconstruction as in Bhandarkar & Chirravuri [9], Job Shop Scheduling and Travelling Salesman Problem as in Ram *et al* [121] and engineering problems as in Leite & Topping [96] and Gallego *et al* [57].

Despite all its good and promising features, one has to admit that the computing time requirement is undoubtedly a critical factor in the economic evaluation of the utility of an SA algorithm in the real world industrial problems application. In order to minimize the effect of this drawback, a promising research direction is the

parallelization of the algorithm, which involves carrying out several calculations simultaneously for its realization [96, 1, 121, 56]. Parallelization of SA though sounds easy, however, should not in any way be thought of, as a simple or trivial task.

The cardinal requirement for parallelizing SA is that such process should be carried out so as not to affect or alter the typical sequential nature (Markov chain) of the algorithm; this is due to the fact that, SA processes a sequence of trials in which the probability of an outcome of a given trial depends only on the outcome of the incumbent (current) trial, and does not in any way depend on the trials in the sequence that came prior to the incumbent trial. There are basically two distinct methods used in implementing parallel SA that were suggested soon after the invention of SA. Dreo *et al* [39] argues, the distinction between these two methods still remains very relevant in modern day optimization problems. These two methods include *Division* Algorithm and *Clustering* Algorithm.

3.5.2.1 The Division Algorithm

The *Division* algorithm allows for implementing several Markov chain computations in parallel using a sizeable number (say M_p) of elementary processors. If we assume a constant number of trials, say, N, then each of the M_p processors is responsible for performing N/M_p trials in the M_p various sub-chains. In order to preserve the main characteristic of the SA algorithm at each temperature level, when all processors finish processing their individual tasks, the incumbent optimal solutions are sent to the master node ($p = 0$), which then selects the best and broadcasts the results to all the other processors ($p = 1, \ldots, M_p - 1$).

The following figure describes how the *Division* algorithm operates

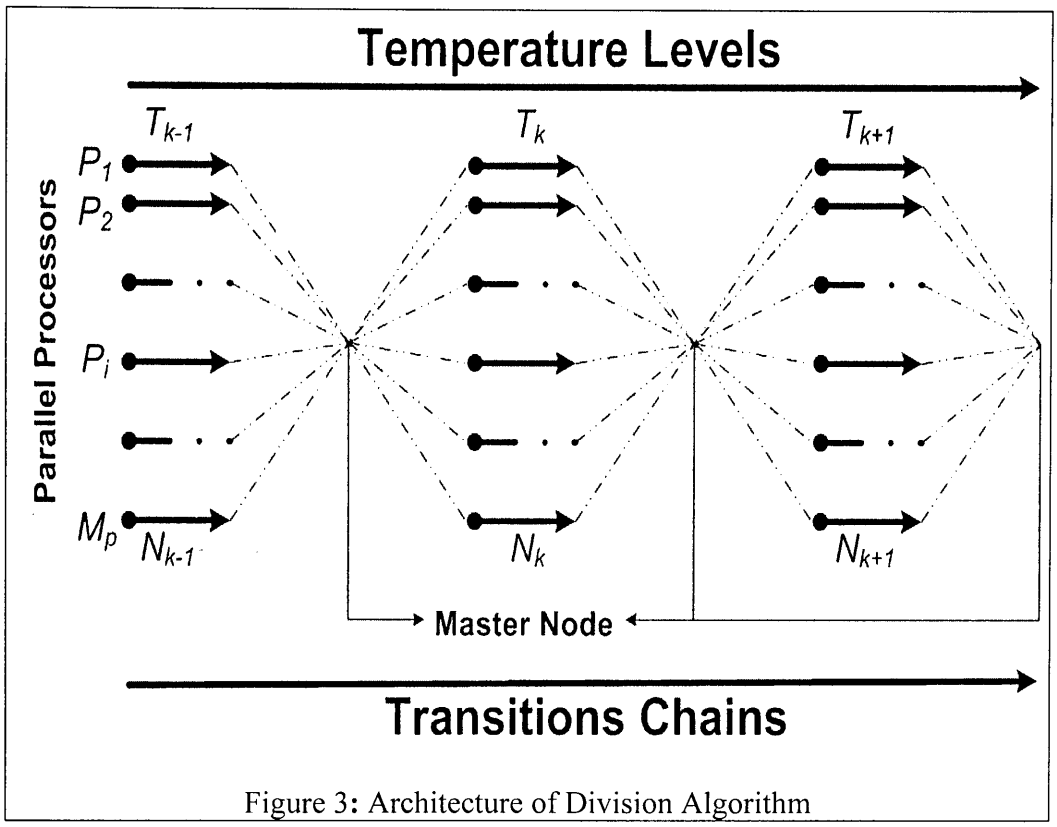


Figure 3: Architecture of Division Algorithm

From Figure 3 above, the horizontal line signifies the evolution of the temperature and the number of trials executed per temperature level; whereas the vertical components signify the M_p processors and each one of them performs its task in a sequence of N/M_p. After all processors finish their assigned tasks, they communicate their "progress" to the master node, which in turn will determine the new best configuration (solution), declare it as the new global incumbent and then makes it available to all processors for onward restart from that point on. It should be noted that, when the master node receives the solutions from the processors it checks whether all the solutions from the processors coincide. This condition is normally satisfied before reaching the minimum temperature level, thereby signifying a faster convergence to the parallel algorithm.

The solution quality obtained through this method relies on the number of processors used in the parallel computation. The main advantage of this method is that, it allows for the division of the total computing time by a factor M_p. However, if a large number of processors is involved, the number of configurations studied by each becomes too small, and this may hinder the system from reaching thermal equilibrium, thereby making convergence towards an optimum unrealistic. One of the remedies to reduce the impact of this problem is to increase the number of trials N per temperature and/or increase the parameter responsible for the rate of temperature cooling, λ. This idea is to make sure that each processor has enough number of trials to simulate near thermal equilibrium conditions as closely as possible.

3.5.2.2 The Clustering Algorithm

Unlike the *Division* algorithm described in the previous section, the *Clustering* algorithm strictly observes the sequential feature of the conventional SA algorithm, in the sense that all the processors involved perform the N trials in a most cooperative fashion by working with the same current solution. The method starts with each of the processors undergoing the processes (i.e. accepted moves, rejected moves, temperature change and so on) involved in the sequential SA algorithm. However, whenever any of the processors accepts a new move, it communicates the resultant solution to other processors which in turn will switch to that solution and all regard it as the new global incumbent and continue the search from there. This process continues until a specified stopping criterion is attained.

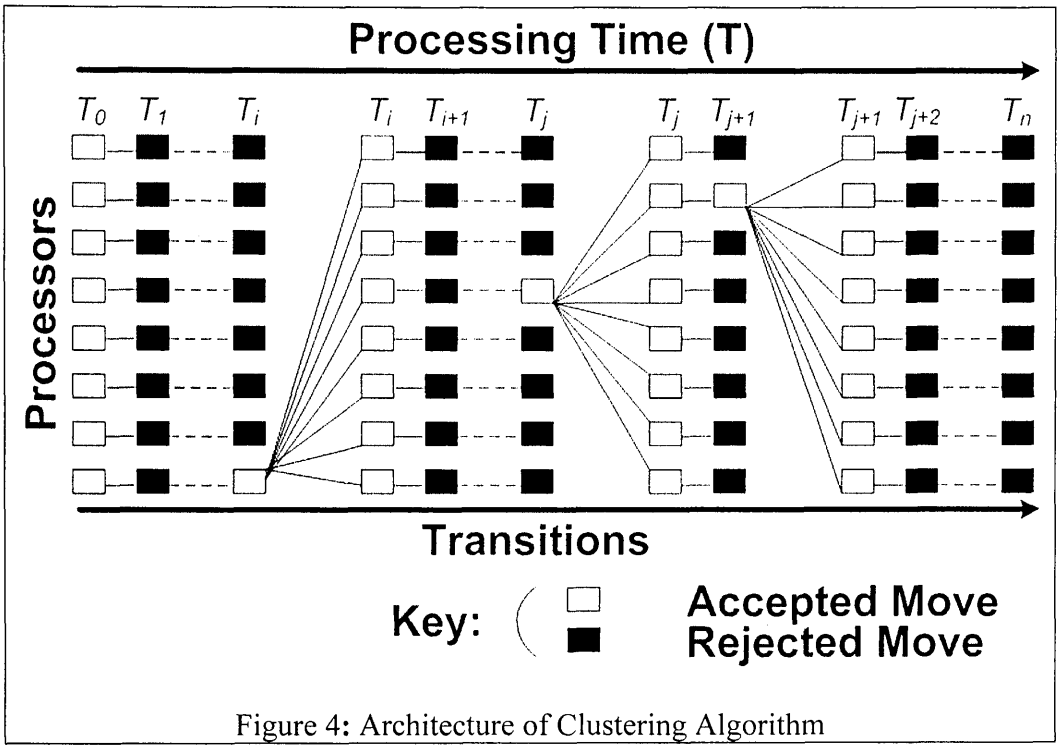


Figure 4: Architecture of Clustering Algorithm

From the description given above, it can easily be inferred that, there is very large frequency of communication among the processors at high temperatures due to the large number of moves accepted at such extreme. On the other hand, the opposite scenario is observed at lower temperatures where very few moves are accepted. Leite & Topping [96] and Monticelli *et al* [113] argue, despite this degree of communication, this algorithm presents a better performance since it doesn't require strong synchronization. Apart from the two techniques of parallelizing an SA algorithm described above, there are other methods as well; an interested reader is hereby referred to Dreo *et al* [39], Monticelli *et al* [113] and Leite & Topping [96].

As for this research, the choice of this method is informed by our drive to explore the capability of parallel SA characterized by improved quality solutions; while at the same time clamping down the time taken by the conventional single-solution SA to arrive at a good solution. We hope to achieve this, by exploiting the *pluses* of SA

(mentioned above) through maintaining a number of candidate solutions (often referred to as processors) that swarm the multidimensional search space at any particular time, thereby maximizing the chances of finding very good (even if not optimum) solution within a shorter time frame.

As already stated before, the parallel implementation of SA is really a non-trivial task; as there are lots of things to take into consideration, ranging from search initialization approaches to different choices of cooling schedules. The method operates in a very similar way to SA, but with several searches going on in parallel, with thus several current solutions at any one time.

3.5.3 Tabu Search (TS)

TS is a metaheuristic designed with the motif of guiding other methods to escape entrapment in a local optimum. It is one of the successfully implemented mathematical optimization metaheuristic techniques. Its main idea was believed to be originally introduced and brought to limelight by Fred Glover in the year 1986 (see Glover and Laguna [63] for details), purposely for solving various combinatorial optimization problems. Two important articles (Glover [61] and [62]) are believed to contain most of the principles upon which the method is based and which are still in use today. However, in the scientific community, some of the principles that guide TS were not well understood in the early nineties, during which there was no such interest in what Dreo *et al* [39] termed as "metaheuristic culture"; as most of the researches conducted in TS, then, used a restricted domain of the said principles, largely limited to *tabu list* and a simple *aspiration condition* [39].

The method is believed to derive its name originally from an 18th century Polynesian word: "*taboo*", and frequently written as "*tabu*"; which is defined according to the Oxford Dictionaries online version [79] as: "... *a social or religious custom prohibiting or restricting a particular practice or forbidding association with a particular person, place, or thing*" or something "... *prohibited or restricted by social custom*". These definitions seem to accord well with the idea behind TS as it makes some decisions prohibitive in order to avoid executing counter-productive course.

The method gained much prominence and attention in the scientific community with the research works conducted by de Werra's team at the Swiss Federal Institute of Technology, Lausanne in the late eighties. Hence, some significant credit should go to de Werra's team for popularization of TS techniques; as their researches: Hertz and de Werra [74], de Werra and Hertz [30], and Hertz and de Werra [75] played a vital role in disseminating the technique in the research community. Despite the growing competition between TS and SA (which was introduced earlier than TS and had to its credit an established convergence theorem), TS-based heuristics were growing in popularity and acceptability especially with some effective and promising results obtained from the works of Taillard [138], [139], [140], and [141].

The method was applied and found to be successful in solving diverse optimization problems including (but not limited to) graph colouring, electronic circuit design, financial analysis, molecular engineering, resource planning, pattern classification, mineral exploration, environmental conservation, biomedical analysis, waste management, flexible manufacturing, quadratic assignment, logistics, telecommunications, energy distribution, space planning, scheduling, character

recognition, to mention but a few. TS, as an extension of classical local search methods provides solutions that are close to optimality; and many regard it as the most effective in tackling difficult optimization problems [11, 139].

Unlike other combinatorial optimization techniques, TS has its origin from concepts used originally in artificial intelligence and not from any physical or biological processes as in the case of SA and/or GA respectively. As such, it possesses some set of principles which when applied in an integrated way will solve a difficult optimization problem in an intelligent manner – a feature which form the base upon which the method is founded. According to Glover and Laguna [63]:

"*Tabu search is based on the premise that problem solving, in order to qualify as intelligent, must incorporate* ***adaptive memory*** *and* ***responsive exploration****... The adaptive memory feature of TS (whose importance is suggested by the analogy of the mountain climber who must analyze current alternatives in relation to previous ascents of similar terrain) allows the implementation of procedures that are capable of searching the solution space economically and effectively... TS contrasts with memoryless designs that heavily rely on semirandom processes that implement a form sampling...*"

TS is sometimes considered as one of the most widespread *single-solution* metaheuristics in use, using "memory" to store information related to the search processes [113]. The method, in comparison to SA and GA, is greedier; as it explores the vastness of the search space in a more aggressive and intelligent fashion than either of the two. Basically, TS begins with an initial configuration (solution) generated

either randomly or in a guided fashion; after which such solution assumes the status of the current solution. During each and every iteration, a neighbourhood structure of the current solution is defined and a *move* to the best solution within such neighbourhood is always accepted (for instance, in a minimization problem, the best configuration refers to the solution with the lowest cost). The so-called best solution may either be chosen based on the *First–Improving* solution criterion or based on complete enumeration of the entire neighbourhood. The term *best* here refers to the solution that improves most the objective function value; however if such *best* doesn't exist within the neighbourhood of the current solution, the *move* leading to a solution that least degrades the objective function value is chosen. In order to avoid getting engulfed in an *intractable* problem, only the most promising neighbours in the neighbourhood of the current solution are evaluated.

Although TS, unlike the stochastic SA, is a deterministic algorithm; it was designed to escape getting entrapped in a local optimum solution. However, TS behaves mostly like a steepest local search algorithm in the sense that, it usually makes an uphill move only when it is entrapped in a local optimum; whereas SA can make such uphill moves at any given time. In executing such an uphill move, TS often permits *moving* to the best candidate solution in the neighbourhood even if it is worse than the current solution (as in the case of SA and other top rated metaheuristic techniques). One of the most important and remarkable features that distinguishes TS from other algorithms, while at the same time playing a vital role in its efficiency across diverse research areas is its ability to develop a mechanism that disallows jumping back or visiting (again) recently encountered solutions for a number of iterations, through the maintainability of what is usually known as a *Tabu List* – which is like a database

containing recently visited solutions or their attributes; thereby preventing certain solutions from reoccurring for a certain number of iterations – called the *size*, *length* or more commonly *tenure* of the tabu list. The elements (*moves attributes*) in the tabu list are added based on the rule widely known as First-In-First-Out (FIFO); which makes it possible for the list to be continuously updated as the algorithm proceeds, so that the move just added to the list can be automatically removed from it after its *tabu tenure* has elapsed.

However, this prohibition of revisiting recently encountered solutions as imposed by tabu list's membership makes TS a little bit restrictive and consequently, may well (occasionally) lessen the efficiency of the method. Moreover, after a while it might be worthwhile to revisit a recently encountered solution from which further searches to some promising direction can begin. Now, in order to improve the method's efficiency and to allow for visiting more promising solutions, an *aspiration criterion* is introduced purposely to override the tabu status of a given solution when it is able to satisfy these (*aspiration*) conditions. With this, it means a TS algorithm can accept non-tabu neighbours as well as those that satisfy the *aspiration criteria* even if they are already declared tabu.

In addition to the design issues related to all *single-solution* metaheuristics (such as the neighbourhood structure and how initial solutions are generated) and other implementation essentials peculiar to TS (such as *tabu list (short-term memory)* and *aspiration criteria* described above), there are also some advanced implementation issues that are introduced into TS to handle issues that focus towards the *intensification* and *diversification* of the search.

- **Intensification** (*medium-term memory*): The main aim of intensification in this method is to exploit and utilize the information concerning the *elite* (best found) solutions, primarily to help in guiding the search towards promising regions of the search space. This set of information is stored in a medium-term memory. The whole idea of intensification involves extracting the common attributes of these elite solutions to further intensify search around solutions with similar features.

- **Diversification** (*long-term memory*): Single-solution metaheuristics (such as TS) are known to possess an intensification power. TS, on the other hand uses long-term memory to encourage diversification. The long-term memory achieves this by forcing the search towards unexplored regions of the search space.

The main components of a TS algorithm's operation are outlined in the following steps and a comprehensive flowchart for those operations follows in Figure 5 below:

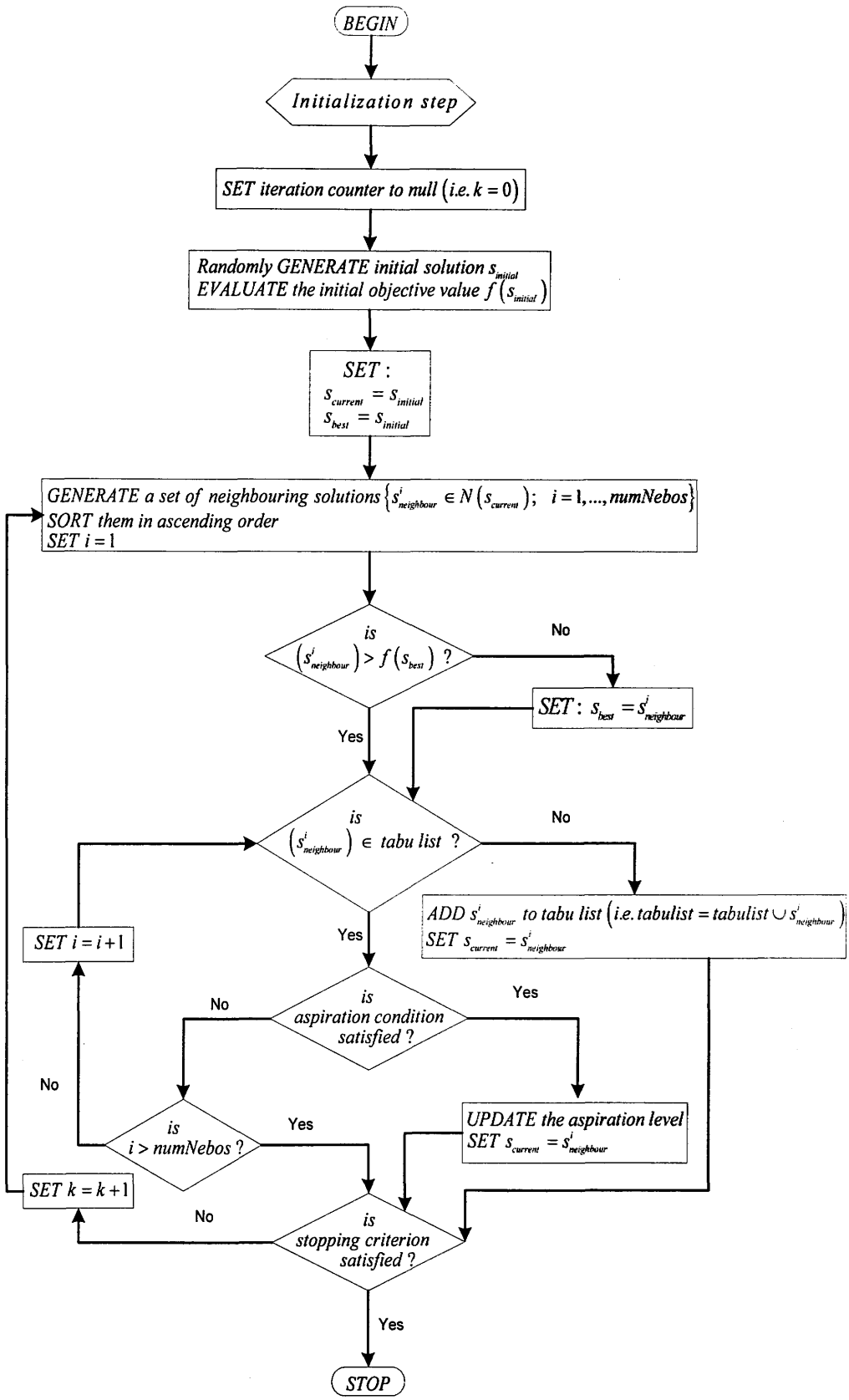


Figure 5: Typical flowchart of a TS algorithm

Step 1:

(a) Generate an initial solution, $s_{initial}$,

(b) Set iteration counter, say k, to null ($k = 0$).

(c) Set the initial solution (generated in step 1(a) above) as the current ($s_{current}$) as well as the best found so far (s_{best}). $\{ s_{current} = s_{initial}; s_{best} = s_{initial} \}$

Step 2:

(a) Randomly generate some set of candidate solutions $\left\{ s^{i}_{neighbour} \in N\left(s_{current}\right);\ i = 1, \ldots, numNebos \right\}$ in the neighbourhood $N(s_{current})$ of the current solution, $s_{current}$.

(b) (Given a minimization problem), sort these neighbouring solutions in ascending order based on their objective function. After sorting, $\left\{ s^{1}_{neighbour} \right\}$ represents the best neighbour in $N(s_{current})$ having lowest objective function value.

Step 3:

Set $i = 1$, $IF\ f\left(s^{i}_{neighbour}\right) > f\left(s_{best}\right)\ THEN\ goto\ step\ 4 - ELSE\ set\ s_{best} = s^{i}_{neighbour}\ THEN\ goto\ step\ 4$

Step 4:

(a) Check the tabu status of $\left\{ s^{i}_{neighbour} \right\}$.

(b) If not a tabu member then add it to the tabu list. $set\ s_{current} = s^{i}_{neighbour}\ goto$ step 7. Otherwise, *goto* step 5.

Step 5:

(a) Check the *aspiration criterion* of $\left\{ s^{i}_{neighbour} \right\}$.

(b) If *aspiration* condition is met, then override its tabu status, update aspiration level and $set\ s_{current} = s^{i}_{neighbour}$ *goto* step 7. Otherwise, increment *i* and *goto* step 6.

Step 6:

If $i > numNebos$ goto Step 7. Otherwise *goto* Step 4.

Step 7:

(a) Check any of the stopping criteria.

(b) If at least one is satisfied, then TERMINATE the search process. Otherwise, increment the iteration counter, *k* and *goto* Step 2.

We decide to implement TS for solving our PSP based on the hope that, much better solutions might emerge from it; as we are satisfied with its ability (as in other researches such as Taillard [138], [139], [140], and [141]) in traversing the search space in an intelligent and guided manner through maintaining a tabu list, upon which very good results are normally obtained. Moreover, TS (in comparison to SA) has less number of parameters to deal with. For more details on TS algorithms, any interested reader should consult Glover [60, 61, 62], Glover and Laguna [63], Hertz and de Werra [74, 75], de Werra and Hertz [30], and Talbi [142].

3.5.4 Genetic Algorithms (GA)

Genetic Algorithm, often abbreviated simply as GA, can be defined as "... *a probabilistic search algorithm that iteratively transforms a set (called a population) of mathematical objects (typically fixed-length binary character strings), each with an associated fitness value, into a new population of offspring objects using the Darwinian principle of natural selection and using operations that are patterned after naturally occurring genetic operations, such as crossover (sexual recombination) and mutation*" [92].

GA was first introduced by Holland [78] based on the natural evolutionary processes (natural selection and genetics) seen in biological organisms, and the method was later made popular by one of Holland's students – David Goldberg who solved an interesting and difficult optimization problem in gas-pipeline transmission control [65]. In evolution processes, some people believe that, populations of individuals evolve in line with Charles Darwin's [29] principles of natural selection and *survival of the fittest* strategy in nature. Fitter individuals adapt more successfully to their natural environment, and consequently stand a better chance of surviving and reproducing than their weaker counterparts which will eventually be eliminated from the population. This concept of *survival of the fittest* implies that the genes from highly fit individuals will spread to an increasing number of individuals in successive generations; as good traits of highly fit parents will tend to produce fitter offspring.

The processes described above are simulated by GA; as its search begins with an initial population of individuals containing constant number of chromosomes (generated either randomly or systematically) and then iteratively applying genetic

operators (such as selection, crossover and mutation) in each reproduction stage. GA is a method applied and found capable of solving an extremely wide range of problems. According to Michalewicz [108] and Coley [24], there are quite a large number of complex optimization problems in which GA was applied successfully; these include jobshop scheduling, image processing, adaptive control, wire routing, game playing, cognitive modelling, TSP, spacecraft trajectories, optimal control problems, aeronautics, robotics, transportation problems, water networks, database query optimization, laser technology, analysis of time series, aesthetics, medicine, very large scale integration (VLSI), solid-state physics, facial recognition and many more.

Conventional optimization methods normally begin with a single candidate solution and the search for an optimal solution continues repetitively by applying some heuristics. On the other hand, GA approach is based on using a population of candidate solutions concurrently searching different areas of the solution space in an adaptive manner. GA, quite often allows for precise modelling of an optimization problem without necessarily having an explicit objective function. Moreover, in situation where the objective function is available, it doesn't have to be differentiable.

GA operates on a population of individuals, in which each is a potential solution to the given problem. The canonical GA is encoded as a fixed-length binary string; however, other encoding methods (including a real-valued encoding) have also been used in other researches across diverse areas. These representations serve as an analogy with the actual chromosome in a biological organism. In solving an optimization problem, an individual member of the population is encoded into *chromosome* representing a candidate solution to the given problem. As the algorithm continues, the population of

individuals evolves through sequential and repetitive application of three important *genetic* operators, namely: *selection*, *crossover* and *mutation*. Whenever, one or more of the individuals has at least one of the above mentioned operators acted upon it, it is called a *parent*; while the resultant individuals are often regarded as *offspring*. Therefore, if two operators are applied in succession, the *offspring* produced by the first operation becomes *parent* to the *offspring* generated after the second operation. The new generation of individuals (offspring) is normally obtained at the end of each iteration upon which each one among them has his fitness evaluated based on the objective function value; highly fit solutions (individuals) are made to be more opportune to reproduce by exchanging their genetic features with other fit solutions through *crossover*. This process is believed to result in new *offspring* solutions that combine the genetic traits of their *crossed* parents. The *mutation* operator is often applied after crossover by perturbing some genes in a chromosome. The offspring may either replace weaker individuals (*steady-state approach*) or the whole population (*generational approach*). The *evaluation-selection-crossover-mutation* cycle is repeated until an acceptable solution that best optimize (minimizes/maximizes) a given objective is returned as the ideal solution or until a given termination criteria is satisfied.

GA, although belonging to a class of probabilistic algorithms, operates differently from other stochastic algorithms, since they combine elements of *guided* search on one hand and those of *random* search on the other. Another advantage with genetic-based methods (such as GA) is their ability to perform a multi-directional search by constantly maintaining a population of potential solutions throughout the search

process; unlike other methods who process a single solution for the entire search space.

Before a successful implementation of GA to a particular problem is achieved, there are several issues that have to be decided, including: the method of representation, way of exchanging information between individuals, how to apply the concept of mutation, the size of the population and the termination criteria. We are going to discuss them in what follows:

(i) *Population size*: This refers to the total number of individuals that an algorithm begins with, carries along and maintained throughout the search history. These are synonymous to *particles* and *parallel solutions* in PSO and parallel SA implementations respectively. There is no optimal population size suitable for all problems, but rather, it is problem-dependent.

(ii) *Generations*: This is synonymous to the total number of iterations in other search methods.

(iii) *Genetic Operators*: A typical GA uses three to four basic operators: *selection, crossover, mutation* and *elitism* to direct the population of individuals towards convergence to a global optimum. These operators are discussed below:

(a) *Selection*: It is aimed at pressurizing the population in a manner similar to that of natural selection obtainable in biological systems. It is this operator that ensures the extinction of the *poorly-performed* individuals, while giving

the better ones a greater chance of promoting their information to the next generation. It is also responsible for determining the convergence characteristics of the algorithm. There are quite a number of selection schemes in use including: *Tournament*, *Truncation*, *Linear Ranking*, *Exponential Ranking*, *Elitist* and the most popular *Proportional selection* [28].

(b) *Crossover*: After *selecting* the parents, there is the need to recombine them to produce offspring for the next generation – a process called *crossover*. It is one of the only two variation operators in GA implementation. Reeves and Rowe [124] explained *crossover* as a process of "replacing some of the alleles in one parent by alleles of the corresponding genes of the other." This operator allows individuals to exchange information in similar manner obtainable in sexual reproduction found in natural organisms. The canonical GA uses *one-point* crossover in which two offspring are produced by two parents after swapping all their alleles to the right of a chosen single locus (point). Typical example of a binary representation's one-point crossover is depicted in the following figure:

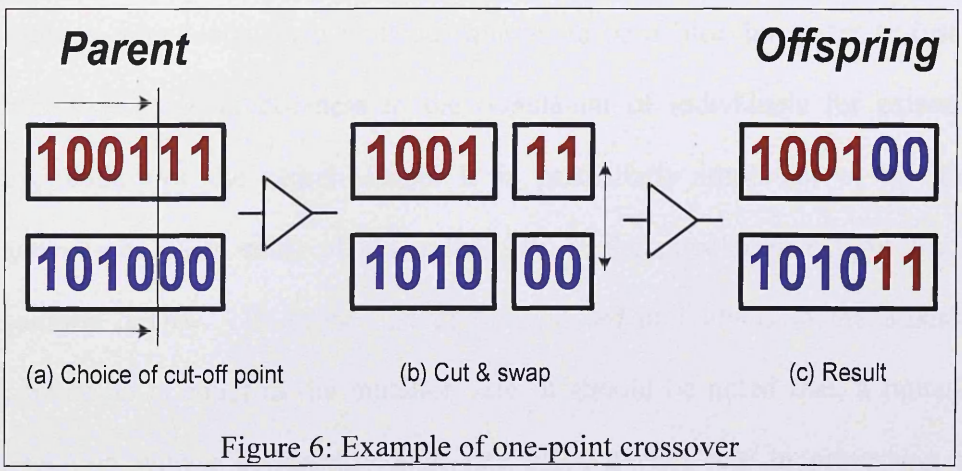


Figure 6: Example of one-point crossover

The crossover process depicted in the above figure produces two offspring

from two parents. However, if only one of the offspring is needed in the algorithm employed, it can be chosen at random from the pair and the other should be thrown away.

There are also some other types of crossover such as *uniform* and *n-point* crossovers; it should be noted at this point that, although more than two crossover points give the algorithm better exploration ability, it often leads to very disruptive configurations.

(c) *Mutation*: This is the other variation operator next only to the crossover operation. In a natural setting, many processes can cause mutation, the simplest being an error in replication. This operator is meant to "*keep the pot boiling*" by modifying an individual's configuration randomly to generate a new offspring that will replace it. Coley [24] argues that, mutation is responsible for maintaining the genetic diversity of the population by preserving the diversity embodied in the initial population, as it is used to stochastically change the value of an allele within an individual chromosome. By so doing, it is believed there is a tendency for a mutated solution to be a little bit better or just to introduce some randomness to the population of individuals for extensive exploration of the search space. It is particularly important at the final generations when most of the individuals in the population exhibit similar solution quality. The proportion of the *mutated* individuals in the offspring population is equal to the mutation rate. It should be noted that, a mutation operation with a sufficiently high rate plays a vital role in preserving the

diversity of the population which is, of course, useful for an efficient exploration of the search space.

For binary encoding, mutation can be carried out by randomly flipping bits with a very small probability. As for real-valued encoding, this can be achieved by random replacement with another random value. Another possibility is by adding/subtracting (or even multiplying by) a random (such as uniformly or normally distributed) amount [28]. An example of binary encoding mutation is:

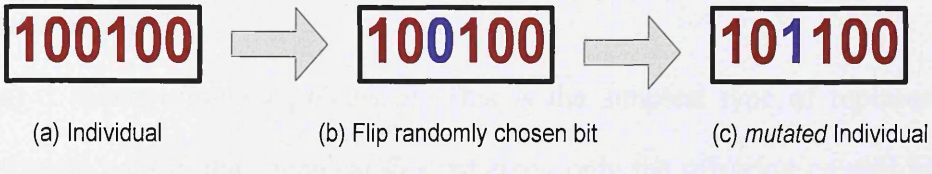


Figure 7: Example of bit-flipping (mutation) operation

(d) *Elitism*: There is no guarantee whatsoever that the *fitness-proportional* (roulette-wheel) or any type of selection method would include even the fittest individual (since the entire selection is probabilistic). However, unless the fittest individual is much fitter than any other, it will occasionally **not** be selected to form part of the next generation, and this simply translates to its demise. This regular throwing-away of *elite* member of the population appears to be, and is indeed, counterproductive. The process of ensuring the propagation of the *elite* member is termed as *elitism*, and requires that not only is the fittest member selected, but a copy of it is not affected by the disruption encountered during *crossover* and/or *mutation* operations. There are quite a number of elitist strategies including one known as $(\mu + \lambda)$–ES which allows for systematic copying of the best parents in the current generation to the population of the next generation. Or if it happens that, the best individual in

the current generation is better (due to the effect of variation or selection operators) than the best in the next generation, then it will be copied to the next generation by simply replacing the worst individual with the lowest fitness.

(iv) *Population replacement*: After the first three or all of the above four outlined genetic operators have been applied to a population; a new population of individuals will have been formed. In GA, the new population of offspring can either replace the whole population (*Generational* approach) or as soon as a new child is generated (*steady-state* approach).

(a) *Generational replacement*: This is the simplest type of replacement strategy used in the canonical GA, whereby only the offspring created in the current generation will form the population of parents in the next generation.

(b) *Steady-state replacement*: This replacement strategy allows for a small number of offspring to be created in each generation purposely to replace equal number of parents in the next generation. This strategy is particularly useful when the solution representation is distributed on several individuals, possibly the entire population. This strategy, by losing a small number of individuals does not disturb the solutions excessively and thus they evolve gradually.

A typical GA begins with a population of stochastically generated individuals which are declared the first generation of parents; their individual fitnesses are then evaluated. Several individuals are then randomly chosen from this incumbent population (based on their fitness values) in order to undergo a modification (recombination and mutation) to form a new population. The new population

undergoes similar processes in the next generation as did the previous one until attaining a maximum number of generations or a satisfactory solution is obtained. Figure 8 shows a flowchart showing the operations executed in a typical GA implementation; and the following pseudocode illustrates the outline of processes involved in a basic GA algorithm.

Outline of the Basic GA

1. [**Start**] – Randomly generate an initial population of n individuals (chromosomes), with each individual serving as a candidate solution to the problem.
2. [**Evaluate Fitness**] – Evaluate the fitness of each individual in the population.
3. [**New Population**] – Create a new population of individuals by executing the following steps repetitively until the desired population size is attained. The steps are:

   (i) [***Selection***] – Select two parent chromosomes from the population according to some defined selection criteria.

   (ii) [***Crossover***] – With some (crossover) probability, perform a crossover operation for the selected parents to form new offspring (children). If crossover operation is not executed, the offspring remain as exact copies of their parents.

   (iii) [***Mutation***] – With some (mutation) probability, mutate an offspring at some chosen point of interest.
4. [**Replace**] – Let the newly generated population replace the old one for further run of the algorithm.
5. [**Test**] – Check if the stopping condition is satisfied, then terminate and return the best solution found; otherwise go to step 6.

6. [**Loop**] – Go to step 2.

GA, according to Haupt & Haupt [70] has lots of advantages over other search techniques which includes its ability to:

(i) Optimize difficult optimization problems with both continuous and discrete variables

(ii) Perform well without the requirement of any derivative information

(iii) Deal with a sizeable number of variables

(iv) Suit well with parallel computers

(v) Provide a list of good solutions, and not just a single solution

(vi) Work with numerically generated data, experimental data, or analytical functions.

However, the sizeable number of solutions that gives the GA its power is also one of its major disadvantages when it comes to speed on a serial computer – since the fitness of each of them has to be evaluated. GA's unique feature of maintaining multiple number of *best* solutions during the course of execution distinguishes it from other (local search and evolutionary) algorithms; and is what we believe gives it an edge over them especially in tackling difficult optimization problems. What informed our decision in this research to choose GA (as one of the optimization techniques) follows from our belief and hope of getting very good and/or near-optimal solutions to our newly formulated PSP. For fuller detail on GA, an interested reader is referred to Dreo *et al* [39], Holland [78], Michalewicz [108], Reeves and Rowe [124], Coley [24], Talbi [142], Da Silva and Falcao [28], Koza [92], Goldberg [65], and Haupt & Haupt [70].

Figure 8: Typical GA operations' flowchart

3.5.5 Particle Swarm Optimization (PSO)

There are quite a number of biological organisms (such as Bees, Ants, Birds, Fish and lots more) that behave as a swarm especially when searching for food or when avoiding an attacking predator. This amazing behaviour has been a focus of many artificial life researches purposely to study, examine and reconfigure such swarm's behaviour inside a computer. For instance, Reynolds [125] developed a swarm model – *boid*, capable of generating animations of complex swarm behaviour using computer graphics. Boyd and Richerson [12] studied human beings' decision process and consequently developed the concept of *individual learning and cultural transmission.* Their studies revealed that, humans make decisions based on their personal experiences as well as other peoples' experiences. Moreover, in the early years of the 1990s, a dawn of new optimization techniques that explore an analogy of swarm behaviour of natural creatures began. Dorigo and Di Caro [36] introduced ACO based on the life style of one of the so-called social insects – the *Ant.* In ACO, each individual (ant) implicitly shares some vital information with other individuals by depositing its *pheromones* trails. Eberhart and Kennedy introduced PSO based on the behaviour seen in the swarms of birds and schools of fish. The collection of researches that involves swarm behaviour is generally regarded as *swarm intelligence* [89]; and of course, PSO is one of the constituent techniques in *swarm intelligence.*

PSO (in relation to SA, TS and GA), is a newly-developed population-based swarm intelligence optimization techniques originally proposed by Russell Eberhart and James Kennedy in the early nineties [88]. The PSO (like other Evolutionary Computation family of Algorithms) is non-deterministic and non-gradient based, implying that no information regarding the gradient of the cost function is needed for

the algorithm to function properly. This advantage makes PSO appropriately applicable in optimizing functions where the gradient is computationally challenging to obtain or even unavailable. It can be used to tackle a vast range of optimization problems, such as financial optimization as in Kendall and Su [86], health problems as in Eberhart and Hu [41] and function minimization as in Shi and Eberhart [130, 131].

3.5.5.1 The Origin of PSO

Jacob and Khemka [82] stated that, the term PSO originated when experimenting with algorithms aimed at modelling the birds' flocking behaviour. In the early years of 1990s, there were several algorithms (such as ACO) designed to simulate flocking behaviour of some organisms; however Kennedy and Eberhart were primarily interested in the Frank Heppner's algorithms also known as *Heppner's Birds*. The algorithm exhibit similar results as other algorithms of the time, exhibiting the following features:

- The *birds* must *fly* towards the same direction as the *bird* in the forefront.
- The *birds* should have equal velocities as their neighbours while *flying*.
- The *birds* must not, in anyway, collide with their neighbours while *flying* close to each other.

However, there are some other things that attracted Eberhart and Kennedy before coming up with their model on particle swarms. These include the fact that: the birds seemed to be attracted to a roosting area, as they (birds) hover/fly around in a flock and suddenly one of them flew over the roosting area and eventually landed on the roost. This phenomenon will cause the other birds to land there as well. Kennedy and Eberhert modified this idea and modelled their *particles* to behave in similar fashion

as they hover (fly) over a solution space, they would (almost all of them) eventually *land* on the best solution.

Although, working in synonymous fashion to other population-based search methods through updating the movements of individuals in their respective populations (and eventually leading to obtaining an optimal solution quickly), the PSO was named as such, because it is believed to be motivated and inspired by simulating the social behaviour of some organisms (Bird flocking and fish schooling) [88, 130, 131, 146]. The basic PSO algorithm plays around with population of points, called *particles* in a multi-dimensional search space, in which each of these *particles* serves as a candidate solution to the optimization problem at hand. Each of these *particles* in the entire swarm flying through the hyperspace possesses a position and a velocity as well as an essential reasoning capability of memorizing their own (local) best position and that of their neighbours (global best). The concept of *best* here is relative depending on the kind of optimization problem at hand; if it is a *minimization* problem, then *best* simply refers to a position in which the evaluated value of the objective or cost function is at its minimum, otherwise it refers to a position which returns the most maximum value for the cost or objective function.

Unlike, other population-based evolutionary optimization search techniques, each *particle* in PSO flies through the multi-dimensional search space with a velocity that is continually perturbed according to its own and its companions' flying experience [131]. Furthermore, these *particles* are able to communicate and relate to their neighbours the history of their trajectories and their best positions found so far, and consequently this inter-communication enables them to adjust their own positions and

velocities. The PSO algorithm begins and continues with the population of particles, achieving effective performance through competition and cooperation among the members. It should be noted that, unlike in some evolutionary computational search algorithms, whereby *selection* operation may render some individuals to become extinct as they die out of the population; the PSO is the only evolutionary algorithm that does not implement what is often referred to as the *survival of the fittest* strategy [42].

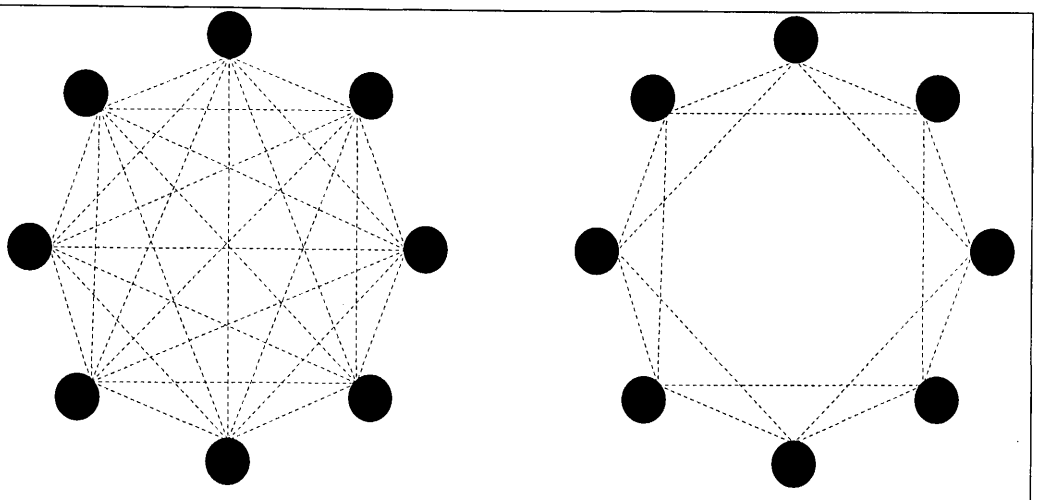
PSO was originally designed for solving continuous optimization problems in which its applications to such class of problems was proposed in Kennedy and Eberhart [88]. Basically, the model consists of a swarm of M particles hovering over an n-dimensional solution space, with each particle, say i ($i = 1, 2, \ldots, M$) serving as a potential solution to the given problem being represented by a vector $\boldsymbol{x}_i$ in the solution space. Each particle also has a *position*, an associated *velocity* and as well possesses an ability to share its trajectory history with other particles in its immediate neighbourhood or the entire swarm; thereby allowing some successful members of the swarm to have some degree of influence over their peers. Each member of the swarm repositions its position $\boldsymbol{x}_i$ towards the global optimum solution based on the history of its most promising and best-ever visited solution – *lbest*(i) *denoted as* $lb_i = (lb_{i1}, lb_{i2}, \ldots, lb_{in})$, and the best solution ever visited by any member in the entire swarm's trajectory history – *gbest denoted as* $gb = (gb_1, gb_2, \ldots, gb_n)$.

3.5.5.2 Particles Neighbourhood

Each particle in the swarm must have some defined neighbours with which to communicate its progress and vice versa. The neighborhood denotes the social influence existing between the particles; and there are quite a number of different possibilities to define such a neighborhood. The two most commonly used neighborhood models include: (a) the *Gbest*, and (b) *Lbest* models.

(a) ***Gbest Model***: In this type of model, each particle considers the entire swarm as neighbors [see Figure 9(a)]. At the expense of robustness, the *gbest* model converges faster; due to its maintainability of a single best solution, often regarded as the *global best particle* across the entire swarm, which acts as an *attractor* having pulling power to attract other particles towards it; and eventually they all converge to its position. Thus, if this global solution is not updated on a regular basis, the swarm may converge prematurely standing the risk of getting trapped in a local optimum.

(b) ***Lbest Model***: Each particle considers a subset of the swarm as its neighbors [see Figure 9(b)]. Local neighborhoods introduce various independent social groups in the swarm, and information between these various subgroups is then communicated back to the entire swarm in some structured fashion. This type of model inhibits premature convergence by maintaining multiple *leaders* (neighborhood best particles). According to van den Bergh [146], this method has two main advantages: (i) it is computationally less costly, and (ii) it assists in promoting diversity and spread of information within the swarm.



(a) **Gbest Model**: Particles are neighbors to one another (Complete graph)

(b) **Lbest Model**: Each particle has only 4 nearest neighbors (noncomplete graph)

Figure 9: Neighborhood topology in PSO

Regardless of the neighborhood topology adopted, an attractor (i.e. *lbest* or *gbest*) is a particle that spearheads other particles towards promising regions of the search space. Suppose we have a swarm of particles whose size is n, each and every member of the swarm is viewed as an object with lots of characteristics. These features or characteristics can be represented by the following symbols:

$\boldsymbol{x}_i$: This vector stores the current position (location) of particle i in the search space.

$\boldsymbol{v}_i$: This vector stores the velocity which particle i travels with, and

$\boldsymbol{lb}_i$: This vector stores the best position ever visited by particle i in its entire search trajectories.

The PSO operates like *cellular automata*, due to the fact that the particle update is executed in parallel and each new value is dependent upon the previous value and its neighborhood. At the instance of each algorithmic iteration, a particle flies from one point to another in the solution space, while at the same time undergoing the following three update operations:

(i) **Velocity update**: The velocity of a particle defines the direction and amount of change (distance) to be applied to a given particle.

$$v_{ij}^{k+1} = v_{ij}^{k} + C_1 R_{1,j}^{k}\left[lb_{ij}^{k} - x_{ij}^{k}\right] + C_2 R_{2,j}^{k}\left[gb_{j}^{k} - x_{ij}^{k}\right] \quad - \quad 3.5.5(a)$$

Where k denotes a unit pseudo time increment (iteration number), R_1 and R_2 are randomly generated and uniformly distributed values in the interval [0, 1] purposely to introduce some stochastic effects in the algorithm. The constants C_1 and C_2 (collectively known as *acceleration coefficients*) are the learning factors influencing the maximum *step size* a given particle can take in a single iteration; additionally, they also scale the values of R_1 and R_2. The velocity update step is handled separately for each dimension j, so that v_{ij} now denotes the velocity vector associated with particle i in the j^{th} dimension. It is easily observable from the above equation that, as C_2 (the social learning factor) aims at regulating the maximum step size in the direction of the global best particle's position; C_1 (the cognitive learning factor), on the other hand, regulates the maximum step size towards the personal best position of the particle in view. In order to reduce the possibility of any particle leaving the search space, the value of v_{ij} is restricted to lie within the interval $[-v_{min}, v_{max}]$.

Shi and Eberhart [130] suggested a modified velocity update mechanism by introducing what they referred to as *inertia weight*, often denoted by w in the velocity update equation given above. The inertia weight, w, restricts the influence of the previous velocity on the current one; in which larger values signify higher influence and lower values means lower influence.

This *inertia factor*, w, has a similar effect in PSO as does the *temperature* in SA; as it serves as a *trade-off* between diversification and intensification during the search. Thus, large inertia weight promotes global exploration of the whole search space, while a smaller one encourages intensifying the search around the current region. The modified velocity update equation (with inertia weight) is now given by:

$$v_{ij}^{k+1} = wv_{ij}^{k} + C_1 R_{1,j}^{k}\left[lb_{ij}^{k} - x_{ij}^{k}\right] + C_2 R_{2,j}^{k}\left[gb_{j}^{k} - x_{ij}^{k}\right] \quad - \quad 3.5.5(b)$$

The original PSO implementation used an inertia weight value of $w = 1$. However, other researchers suggested different values. In order to briefly illustrate the effect of w; let us now set $C_1 = C_2 = 0$. A value of $w < 1.0$ will make the particle decelerate slowly until its velocity reaches zero; on the other hand, a $w > 1.0$ will make the particle accelerate up to a maximum velocity, v_{max}, given that, the particle started its search space trajectory from non-zero velocity. However, based on the results obtained by Shi and Eberhart [130], an inertia weight value close to unity is preferred.

(ii) **Position update**: Each particle will update its coordinates on the solution space using the new particle's velocity updated vector, and this can be achieved by setting:

$$x_{ij}^{k+1} = x_{ij}^{k} + v_{ij}^{k+1} \quad - \quad 3.5.5(c)$$

(iii) **Best found solution update**: Each particle updates its best local solution when a better solution is found as the search progresses. Suppose for a minimization problem we denote by f, the objective function to be minimized; then the personal best position of a particle can be updated by the following pair of equations:

$$lb_i^{k+1} = \begin{cases} lb_i^k & if\ f\left(x_i^{k+1}\right) \geq f\left(lb_i^k\right) \\ x_i^{k+1} & if\ f\left(x_i^{k+1}\right) < f\left(lb_i^k\right) \end{cases} \quad - \quad 3.5.5(d)$$

Moreover, the global best solution found by the entire swarm will be updated as follows:

$$gb^{k+1} = \begin{cases} gb^k & if\ f\left(x_i^{k+1}\right) \geq f\left(gb^k\right) \\ x_i^{k+1} & if\ f\left(x_i^{k+1}\right) < f\left(gb^k\right) \end{cases} \quad - \quad 3.5.5(e)$$

The above three steps of velocity and positions update in conjunction with particles' fitnesses calculation and evaluation will be continuously executed in a repetitive manner until a desired convergence or stopping criteria is satisfied; which is often the maximum change in best fitness should be smaller than a specified threshold value for a specified number of iterations.

In brief, a basic PSO algorithm begins with some randomly generated *particles* (candidate solutions), among which one (based on its fitness) is identified and tagged as the global best (*gbest*) solution. All other particles in the swarm will then accelerate towards the direction of this (*gbest*) particle, while at the same time being *drawn* toward the direction of the best solution they ever discovered in their search trajectories' history. Partaking in this phenomenon will occasionally make the particles overshoot their target, consequently allowing them to explore the search space beyond the current best particles; while at the same time having the opportunity to discover better solutions *en route*, in which case the other particles will change their direction heading towards the newly found best solution. Due to the fact that most

functions have some continuity, there are chances that a good solution will be surrounded by other good or better solutions; and the fact that these particles approach the current best from different angles in the search space, there is a strong likelihood that these (better) neighbouring solutions will be discovered by some of the particles. The following figure depicts the velocity and position updates of a particle in a typical PSO implementation:

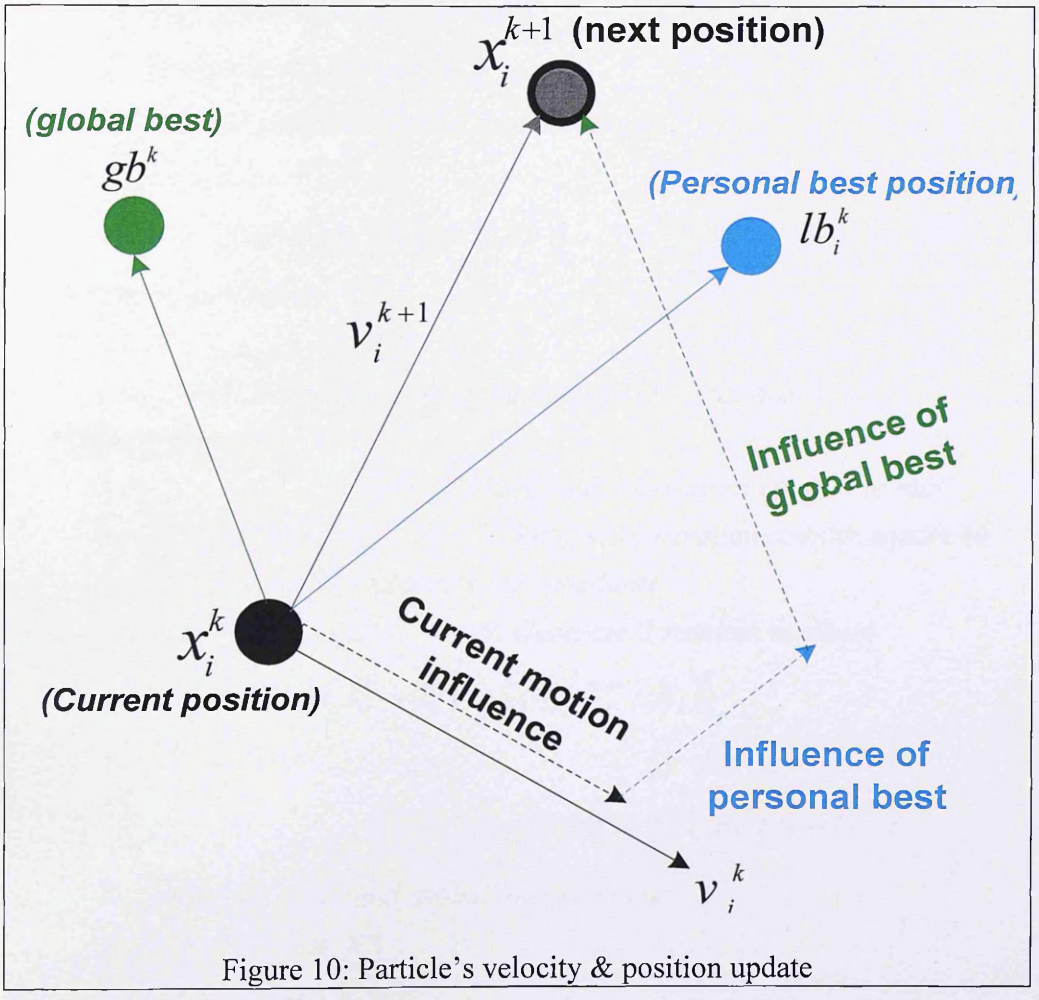


Figure 10: Particle's velocity & position update

The following pseudocode depicted on Figure 11 describes how we implemented our unconstrained PSP using PSO

Particle Swarm Optimization $\left(inputs: p, C_1, C_2, M, X_{ij}; output: X_j^{gb}\right)$

Begin

Choose C_1, C_2 *% Decide on the values of acceleration coefficients*

% Initialize particles' size (p)*, dimensions* (n) *and Number of iterations* (M)

Choose p, n, M

% Initialize p particles' positions and n dimensions

Initial solution $X_{ij} = rand \in (0,\ 1);\ \forall i = 1 \ldots p\ ;\ j\ =\ 1 \ldots n$

Initial velocities $V_{ij} = rand \in (0,1);\ \ \forall i = 1 \ldots p\ ;\ j\ =\ 1 \ldots n$

% Evaluate Initial Particles' fitnesses

$f\left(X_i\right) = evaluated\_fitness\left(X_{ij}\right)$

% Initialize the local and global best solutions

$f\left(lb_i\right) = f\left(X_i\right);\ X_{ij}^{lb}\ =\ X_{ij}$

$f\left(gbest\right) = \left\{Min\left(f\left(lb_i\right) \mid \forall i\ =\ 1 \ldots p\right)\right\}$

$minimumfitnessindex\ =\ i$

$X_j^{gb}\ =\ X_{minimumfitnessindex\ j}^{lb} \forall j\ =\ 1 \ldots n$

% Iterate until convergence or finite number of iterations

Repeat Procedure

$k = 0$ *% Initialize iterations counter to null*

For $k = 1$ *through* M *% Loop until iteration counter equals M*

% update particles' velocities and positions

$R_1 = R_2 = rand \in (0,1)$ *% Generate 2 random numbers*

$V_{ij} = \left\{wV_{ij} + C_1R_1\left(X_{ij}^{lb} - X_{ij}\right) + C_2R_2\left(X_j^{gb} - X_{ij}\right)\right\}$

$X_{ij} = X_{ij} + V_{ij}$

$\sum_{i,j=1}^{p,n} X_{ij} = 1$ *% Normalize to ensure feasibility*

% update the local and global best solutions

Update $f\left(lb_i\right)\ \&\ X_{ij}^{lb}$

Update $f\left(gbest\right)\ \&\ X_j^{gb}$

End (For)

Return *the global best solution* $\subset \left(f\left(gbest\right)\ \&\ X_j^{gb}\right)$

End

Figure 11: Pseudocode of a typical PSO implementation

Our research, by designing and implementing PSO, is aimed to exploit the PSO's capability to obtain near-optimal solutions for our PSP.

3.5.6 SWarm ANnealing (SWAN)

This algorithm was designed as a hybrid of PSO and SA and derives its name by combining two selected words from the names of the constituent algorithms. Due to the diversity and quality of solutions returned by different *particles* (candidate solutions) in PSO implementation and the ability of SA to obtain a very good solution by *intensification* especially at lower temperatures; by hybridizing the two we hope the algorithm will be able to combine these main heuristics' desirable features (*Diversification & Intensification*) from both PSO and SA.

Eglese [43] argues that the hybridization of SA with another method (such as PSO) can be done in two different ways. First, the hybridization should be in such a way that either the other method is used to obtain a good initial solution after which an obtained solution would then be passed to SA for improvement or the other way round. In our implementation, we decided to adopt the first approach, by passing the (global) best solution returned by PSO to SA optimizer for further improvement.

Similar implementation (of PSO and SA hybrids) exists in the literature [147]; however, we want to clearly state here that, our implementation is significantly different from what is contained therein. This is because in Wang and Li [147] each of the PSO generated particles (candidate solutions) is subjected to the SA optimizer by undergoing all the processes involved in SA after which the particle with the best solution is declared global best (*gbest*) and all other particles' positions are then

updated according to the PSO update scheme. This process continues in similar fashion until convergence or the given desired stopping criterion is attained. But in implementing our SWAN algorithm, all the particles undergo all the processes (velocity, particles' positions, local best solutions and global best solution updates) involved in a typical PSO technique until convergence; the *gbest* solution is then passed to the SA optimizer as a starting solution for further improvement until a given stopping criterion is attained. By implementing the algorithm in such a manner we expect it would be able to produce results of superior quality than either of the two techniques when implemented separately.

For this method (SWAN) to be considered as a hybrid of both algorithms, it must combine their collective parameters. Furthermore, the tuning of search parameters as far as this method is concerned will be a little bit more challenging, since the parameter settings that were found to work well with our PSO and/or SA algorithm might not be found to work well with the hybrid. So it needs further parameter fine tuning. The following *pseudocode* shows how our SWAN algorithm operates:

Swarm Annealing $\left(\textbf{\textit{inputs}} : p, C_1, C_2, M, X_{ij}, T, \alpha, N;\ \textbf{\textit{output}} : X_j^{gb}\right)$
Begin
  Particle Swarm Optimization $\left(inputs : p, C_1, C_2, M, X_{ij}; outputs : gbest^{PSO}\right)$
  $s = gbest^{PSO}$
  Simulated Annealing $\left(inputs : s, T, \alpha, N; outputs : X_j^{gb}\right)$
End

Figure 12: Implementation of a SWAN for PSP

Where the parameters: p, C_1, C_2, M, T, α, and N are the number of particles, cognitive factor, social factor, number of iterations, maximum temperature value, cooling rate and size of the Markov chain respectively.

Due to the presence of numerous parameters in our SWAN algorithm, there is the need for patience in trying several parameter combinations before finally settling on the one that seems to produce a reasonably good solution in an acceptable time frame. When this is done, we expect our SWAN algorithm to perform better than either the PSO or SA when implemented separately due to its ability to explore the features of both algorithms in a single experimental trial.

4.0 Unconstrained PSP Implementation

4.1 Practical Implementation of PSP: The Unconstrained case

In order to test our algorithms' performances and robustness, we decided to run (for each of them) an unconstrained formulation of the PSP involving 2 different datasets as described in section 4.2 below. For each of the datasets, each of the 6 algorithms (described in section 3.5 above) was run 50 times and a mean value of the 200 generated solutions (portfolios) on the efficient frontier was then computed. These results were then compared with the ones obtained by solving the same problem instance using a standard quadratic (nonlinear) programming solver (CPLEX 11.2) invoked by a script in AMPL modelling language coded for such purpose.

4.2 Datasets used for the research

Although, there are several datasets available online to test our proposed algorithms; we decided, for the purpose of this research, to test our algorithms based on just two. The first one is a weekly stock price data from March 1992 to September 1997 for the *Hang Seng* (Hong Kong) capital market index made publicly available at the OR Library [117]. The dataset contains the input vectors (covariance matrix and return vectors for 31 *ID-concealed* stocks) needed for solving the PSP.

The second set of test data is a freshly downloaded weekly stock price data from FTSE100 (UK) capital market index. Stocks with missing values were disregarded and we ended up with 78 stocks; and for each we obtained 262 weekly price data from January 2004 to January 2009.

Although two may seem a very small number of data sets, a very wide range of experiments has been undertaken for these data sets, using a variety of methods, with varying parameters, and with a wide variety of constraints. This means that the total number of results produced is large and comprehensive.

We then decided to compute logarithmic $\left[\ln\left(P_t / P_{t-1}\right)\right]$ weekly returns; upon which we then compute their respective expected returns and covariance matrix which eventually serve as input vectors to our optimization problem. P_t stands for an asset's price at time t.

4.3 Algorithmic Implementation Details: Parameter choice decisions

This section is aimed at describing how our algorithms were implemented in the unconstrained formulation. Several issues play a significant role in achieving proper and successful implementation of metaheuristic algorithms in solving any difficult optimization problem. These issues can be viewed from basically two challenging perspectives, namely the *generic* and *problem-specific* choices [43, 65]. Take SA for instance, the *generic* choices deal with making specific statements on the acceptance probability functions together with making choices on cooling schedules; as for *problem-specific* choices, this deals with making decisions on the solution space, neighbourhood structure, the objective function and possibly the constraints to be satisfied. We are now going to explain the key decisions reached in implementing our solution techniques in relations to the *generic* and *problem-specific* decisions.

4.3.1 Problem-Specific Decisions

Because, all our algorithms are aimed at solving the same instance of the PSP, synonymous decisions and choices were made in relation to those that are *problem-specific.* According to Wright [149], there are issues that any *problem-specific* decisions have to address, and some of these include:

(i) Problem Definition: The PSP was originally modelled on a single-criteria basis, but on the alternative it can be viewed as a multi-criteria (bi-objective) optimization problem, where the portfolio risk (often measured by the variance) is to be minimized while at the same time trying to maximize the portfolio return. Our research adopts the alternative formulation as our cost function follows similar but different fashion as used in Schaerf [129], where the cost related to the violation of the return constraint is combined with the original objective function; and this makes the overall objective function a weighted sum of the return and risk components as follows:

$$\text{Minimize} \left((1-\lambda)\left(\sum_{i=1}^{n} \sum_{j=1}^{n} w_i \, \sigma_{ij} \, w_j \right) + \lambda \left(\left| \sum_{i=1}^{n} w_i \, \bar{r}_i - R_T \right| \right) \right) \quad - \quad 4.3.1$$

Where R_T is the supplied target aimed to be achieved, λ is the penalty for violating the return constraint.

In our implementation we decided to use variance other than the semi-variance we initially settled on due to the lack of necessary data to do this. For instance, the Hang Seng dataset used in this research (available at the OR library [117]) does not provide the values of the (symmetric) semi-covariance matrix needed for computing the semi-variance of any given portfolio.

(ii) *Search Space Definition*: A solution in PSP can be represented by a sequence of n variables, $w_1, w_2, \ldots, w_n$; where each w_i stands for the fraction of portfolio fund invested in asset i (or the actual amount allocated to asset i if an integer variable formulation is modelled).

In this unconstrained formulation, all our algorithms avoid coming across an infeasible solution, hence we adopt an *All- feasible approach* (of constraint handling) where any of the candidate solutions must satisfy all constraints involved at any stage of the search process. This is also the type of approach implemented by Chang *et al* [20].

(iii) *Neighbourhood Definition*: Wright [149] argues that, this stage plays a crucial role in determining the success of any neighbourhood search method. It is the stage that defines how a neighbouring solution can be reached from the current solution, and this can be achieved by initially defining a set of allowable moves (perturbations). Neighbourhood relations, according to di Tollo and Roli [34] can generally be viewed from 2 perspectives:

- Neighbours being generated by modifying weights of some of the assets in the current solution; and
- Neighbours generated by perturbing all the weights of the assets in the current portfolio.

For the unconstrained formulation, our algorithms adopt the second approach, after which all the assets' weight are normalized accordingly.

(iv) *Generation of an initial solution*: The relevance of a rightly chosen initial solution in producing high quality solutions in most neighbourhood search techniques can never be underestimated; and thus should be considered as non-trivial task [4]. Catanas [18] proved that metaheuristics designed (especially) for solving PSP tend to be robust with respect to the right choice of initial solution. Similarly, Wright [149] emphasized the importance of choosing a good initial solution in ensuring a high-quality final solution.

In our research, all our algorithms begin with a randomly generated solution for the first supplied target return; while for the subsequent targets, the near-optimal solution found for the previous portfolio serves as the starting solution and the process continues in this fashion.

(v) *Acceptance Criterion*: Two main variants of acceptance criterion are the *First-accept Local Improvement* and *Best-accept Local Improvement*. In the former the newly generated neighbouring solution can only be accepted if its cost (objective value) is smaller when compared to the cost of the current solution. While in the latter, several neighbouring solutions are generated out of which the one with the smallest cost (objective value) is then accepted as the next current solution. In this study, all our local search related algorithms (except the TS) adopt the second approach.

For TS, the acceptance criterion is implemented differently in the sense that any neighbouring solution generated (based on the second approach outlined above) can only be accepted as the next current solution, if and only if:

(a) The solution is **NOT** already in the tabu list or tabu region.

(b) The solution satisfies an *aspiration condition*, which normally overrides a tabu status. The aspiration criterion is that, the magnitude of the current solution's objective is lower than or equals to the *Best-Ever-Found* objective value *even if it is already in the tabu list or lies within the tabu region* of the current solution.

(vi) ***Stopping Criterion***: Also known as termination criterion; it refers to the condition which must be satisfied before the entire search process comes to an end. Because some of our algorithms (like SA and TS) are single-agent methods; while the rest (GA, PSO, SWAN and parallel SA) are multi-agents techniques; we feel it might not be fair to compare their performance while executing the same stopping criterion. In view of this, we executed a stopping criterion that will make sure that, our single-agent methods are not disadvantaged in favor of their multi-agents counterparts.

First, after several experimental runs were conducted, we found that on average a maximum number of **3000** iterations/generations will be sufficient enough for our multi-agents algorithms to produce a very good solution within a reasonable time span. However, in order to save time, a given search process can terminate if there are 500 consecutive non-improving cost function evaluations or when the absolute difference between the portfolio and target returns is no more than a negligible predetermined threshold value of $\varepsilon = 10^{-10}$.

Mathematically, the stopping criterion for our multi-agents methods is executed as follows:

$$If\left(\left(nonImprovementCount = 500\right) OR\left(Abs\left(portfolioReturn - targetReturn\right) \le 10^{-10}\right)\right) Then$$
$$Terminate\ the\ Search\ Process$$

It should be noted that, after several experimental trials we found that our multi-agent methods can produce solution of very good quality with no more than **50** agents (particles/chromosomes). So in implementing a stopping criteria for single-agent techniques (SA and TS), we took this into consideration. The details will follow in the subsequent sections.

4.3.2 Generic Decision Parameters:

This section discusses the decisions reached in dealing with parameters peculiar to a particular search method.

4.3.2.1 SA

(i) ***Acceptance Probability***: In this research, we adopt the most frequently implemented acceptance probability function also known as the Metropolis acceptance criterion and given by:

$$P(Accept\ solution, S_j) = \begin{cases} 1 & if\ f(S_j) - f(S_i) \le 0 \\ e^{-\left(\frac{f(S_j) - f(S_i)}{T_k}\right)} & otherwise \end{cases}$$

The above probability tells us that, whenever a new neighbouring solution is generated; provided it has a lower or cost equivalent to the current, it will certainly be accepted as the next solution. But on the other hand, if the new neighbouring solution

generated returned a deteriorating cost value, such a solution stand a chance of being accepted or rejected based on the cost difference and the temperature value. This is because; the probabilistic value of the exponential function is compared with a uniformly distributed and a randomly generated number lying in the interval [0, 1]. The new solution is then accepted whenever the value returned by the exponential function is found to be larger than the random number generated.

(i) ***Cooling Schedule***: In our SA implementation, the cooling schedules adopted are composed of: the initial value of the temperature, the cooling rate, the length of Markov chain and the final value of the temperature. The numerical values of these parameters reported in this section were arrived at, after a quite number of experimental runs were conducted and some performance measures observed.

(a) ***The initial temperature****:* In all the conducted experimental runs, we pegged the initial value of the temperature parameter at $T_0 = \mathbf{1.0}$. This decision was reached at, after several simulation runs were conducted and found that such value is more likely to return a very good (and many times optimal) solution.

(b) ***The Cooling Rate****:* The cooling rate often denoted by α, is set in such a way that the temperature *cools* reasonably slowly in order to arrive at a very good solution in a reasonable time frame. Empirical evidence points to the advantages of setting this (α) value to **0.99063**. This value ensured that the temperature was reduced from the desired initial value to the desired final value within a reasonable and acceptable run time.

(c) ***The length of the Markov Chain****:* In order to conform with the suggestion made by Eglese [43] that: Time should not be wasted at larger as well as lower values of temperature; after several experimental runs we arrived at a decision to set the length at fixed value of 9 iterations per temperature value, therefore $N_k = 9$, $k = 0, 1, 2,..$

(d) ***The Final temperature****:* For all our algorithms, the value for this important parameter was pegged at a small value of **0.001** – a value very close to zero, hence simulating a *frozen state* of an annealing algorithm.

Details of experimental results and discussion can be found under the heading: "SA parameter choice decisions" in APPENDIX 2.

(ii) ***Total Number of Neighbours considered***: This refers to the number of candidate solutions around the immediate neighbourhood of the current solution in our local search methods. In order to be fair to our single-agent local search methods (TS & SA), we decided to generate **50** (equal to the number of individuals in GA, processors in parallel SA and particles in PSO & SWAN) neighbours around any given incumbent solution, among which the best is picked as the next incumbent.

4.3.2.2 Parallel SA

This search method operates in similar manner to the SA, the only difference lies in the number of solutions dispersed over entire search space, hence parallel SA. Therefore, all the decisions (*generic* and *problem-specific*) reached in relation to the SA are as well adopted in this algorithm. However, we decided to fix the number of (parallel) processors to 50 in both cases.

Due to the fear of premature convergence in implementing the division algorithm, we decided to implement the clustering algorithm which is proved to perform better than the division algorithm [96, 113]. By doing so, we hope to obtain solutions that are *at least* as good as those obtained by SA.

4.3.2.3 TS

(i) ***Tabu Tenure***: This generally refers to the number of iterations for which a candidate solution remains in tabu (forbidden) state, hence having such a solution as the next current is not allowed. In this research, we set the tabu tenure to a value of $T = 7$.

(ii) ***Tabu Region***: Due to the continuous nature of the variables in PSP and also to avoid getting stuck in a local optimum during the search process; we declared what is known as a *tabu region*. This simply means the immediate region within the reach of the current solution in a single transition in which a move is disallowed. In order to implement this, we compute the *Euclidean distance* between the current and any other candidate neighbouring solution, and if the distance is found to be less than a pre-specified *threshold value* the move is declared tabu, otherwise it is accepted. After several independent runs we arrived at a reasonable threshold value of $\mathbf{10^{-5}}$.

(iii) ***Total Number of Neighbours considered***: As in SA implementation above, the number of neighbours considered is **50**, which is equivalent to the number of chromosomes (GA), particles (PSO & SWAN) and parallel processors (parallel SA).

4.3.2.4 PSO

(i) ***The Acceleration Coefficients***: These are basically the *cognitive* and *social* components' coefficients respectively denoted by C_1 and C_2. They influence the maximum step size a particle takes in a single iteration. In the original implementation of the PSO these values were recommended to be set such that $C_1 = C_2 = 2$. As these are values that are problem-dependent, the best-performed configuration of these values found for the PSP tackled in this research (after several trial runs) are $\mathbf{C_1 = 0.95}$ and $\mathbf{C_2 = 2.955}$.

For further details on how we arrive at such decision, check PSO parameter choice decisions in APPENDIX 2.

(ii) ***The Inertia Weight***: Also known as *Inertia Factor* and often denoted by *w*; is a scaling factor (taking real values) associated with the velocity during the previous time step which results in a new velocity update equation. In some PSO implementations it can be fixed, while in others such as Kendall and Su [86], it was set to be dynamic, typically taking values between 0.4 and 0.9.

In order to allow for proper exploration of a very large area of the search space at the beginning of the simulation runs and to further refine the search at later stage, we decided to adopt the dynamic approach in which the inertia weight initially takes the maximum value of 0.9, and as the search progresses it takes different values within the real interval [**0.4**, **0.9**] up to the point where it takes the minimum value of 0.4.

In order to compute our inertia factor, *w*, as the search progresses; we adopted the following equation as used in Kendall and Su [86]:

$$w = wMax - \left(\frac{wMax - wMin}{maximum\ number\ of\ Iterations} \right) * i^{th}\ iteration\ count$$

Where *wMax* and *wMin* are the maximum and minimum values that *w* can take respectively. For further details on how we justified using the dynamic approach of setting the inertia weight values, we refer an interested reader to a section in APPENDIX 2 entitled: PSO parameter choice decisions.

(iii) ***The velocity update rule***: Deviating a little bit from the original velocity updating strategy discussed in Kennedy and Eberhart [88], we adopt the widely used velocity update rule incorporating an *inertia factor* in the update equation as follows:

$$v_{ij}^{k+1} = wv_{ij}^{k} + C_1 R_{1,j}^{k} \left[lb_{ij} - x_{ij}^{k} \right] + C_2 R_{2,j}^{k} \left[gb_j - x_{ij}^{k} \right]$$

(iv) ***Swarm's position updating strategy***: The particles' position updating rule can either be classified as *synchronous* or *asynchronous* [17]. In the former, a particle's position is updated before evaluating the objective function, while in the latter the objective function is evaluated after the swarm updated its position. As for this research, we decided to implement the *synchronous* method. The particle's update equation is given by:

$$x_{ij}^{k+1} = x_{ij}^{k} + v_{ij}^{k+1}$$

(v) ***Number of particles***: In our PSO implementation, we tested different number of particles as detailed in APPENDIX 2 under the heading: PSO parameter choice decisions. The results presented therein reveals that, particles' size as moderate as **50** is sufficient enough to produce a very good result and many times near-optimal solutions within a reasonable period of time.

4.3.2.5 SWAN

It shouldn't be surprising that our SWAN algorithm combines all the parameters used of both PSO and SA algorithms; as it comes into being as a result of hybridizing the duo. Thus, based on the empirical evidence available in APPENDIX 2 under the heading SWAN parameter choice decisions; the best parameter configurations that are found to work well with SWAN unconstrained PSP implementation includes:

(i) *The Acceleration Coefficients*: The acceleration coefficients that were found to produce a better output for the unconstrained PSP are: $\mathbf{C_1 = 0.95}$ and $\mathbf{C_2 = 2.955}$.

(ii) *The Inertia Factor*: This, as in PSO, was set to take values within **[0.4, 0.9]** inclusive.

(iii) *The velocity updating rule*: This is implemented as in the PSO, where the velocity update rule incorporates an *inertia factor* in the update equation

(iv) *Swarm's position updating strategy*: As in the PSO, we adopt the *synchronous* updating rule.

(v) *Number of particles*: As one of our multi-agents methods, we used the same number of particles as used in PSO implementation above.

(vi) *Acceptance Probability Function*: This is similar to the probability function used in section 4.2.2.1(i) above.

(vii) *Cooling Schedule*: Because the SA part of the SWAN algorithm is meant to fine tune the global solution found by the PSO part; we found that a *cooling rate*, α = **0.99063** and a *length of the Markov Chain*, $\mathbf{N_k = 1}$ was suitable enough to produce a very good result.

(viii) *Total Number of Neighbours considered*: For the SA part of this algorithm, we consider only **50** neighbouring solutions around any current solution.

4.3.2.6 GA

(i) ***Population size***: This refers to the total number of individuals (chromosomes) that participate and continued to be maintained throughout the search history. These are synonymous to *particles* and *parallel solutions* in PSO and parallel SA implementations respectively. During the initial implementation of this algorithm we tried a population size of 100 (more than 3 times the dimension of our smaller dataset), but as we kept on improving it, we found that as few as **50** individuals often provide very competitive solutions; and coincidentally, this tallies with the number of particles and processors in PSO and parallel SA respectively.

(ii) ***Generations***: This is synonymous to the total number of iterations in other search methods. So to keep in tune with other algorithms, we set the total number of generations to complete a cycle at 3000; this was also found to be sufficient enough to provide a very good solution.

(iii) ***Genetic Operators***: A typical GA uses three to four basic operators: *selection, crossover, mutation* and *elitism* to direct the population of individuals towards convergence to a global optimum. These operators are discussed below:

(a) ***Selection***: Although, there are several ways in which this operation can be executed, for this research we found *roulette-wheel selection* approach (which is proportional to the fitness of an individual) more convenient to our

type of problem.

(b) ***Crossover***: For our formulation, because we are dealing with a real-valued encoding, we decided to implement a *one-point* crossover in which two offspring are produced by two parents swapping all the alleles to the right of a chosen single locus (point) and we set the probability of crossover to be 0.95, which means there is about 95% chances that any particular solution will undergo this process.

(c) ***Mutation***: We allocate a 1% chance for conducting mutation in our GA implementation.

(d) ***Elitism***: We decide to always carry fittest individuals amounting to 10% of the entire population size to the next generation as part of our elitism operation.

(iv) ***Population replacement***: As in Chang *et al* [20], we employ a *steady-state* population replacement approach, in which pair of newly *born* children replaces a pair of less-fit members of the old population and the process continues until the desired population size is attained.

4.4 Description of the bi-objective problem implementation

All the designed algorithms were implemented in such a way that, when they are run successfully to the end, they will be able to generate an *approximate* efficient frontier of solutions of portfolios.

Conventionally, our algorithms are able to return a single solution, but we have modified them to behave in a repetitive fashion; so that at the end of any single successful run, they would be able to return a set of number of solutions/points each characterized by return and corresponding risk. When the points are plotted on a risk-return plane, a parabola-like curve of efficient points often referred to as an *efficient frontier* is generated. These points are obtained by in(de)creasing the supplied target return in an equally-spaced manner depending on the total number of points desired to make up the frontier.

First of all, a given target return value is passed as input into an algorithm, which seeks to find assets' weights configuration that would determine a portfolio return that is as close as possible to the supplied target. If a portfolio return that matches the target is found, the algorithm would then try to (in subsequent iterations) find a lower value of portfolio risk without compromising the corresponding portfolio return.

For instance, suppose we want to generate an *approximate* efficient frontier with 100 portfolios in which the initial target (return) is 15% and we want a final target return value of 2% to be achieved. Now by following the rule of Arithmetic Progression (AP), our first term (often denoted by a) is 15% while the (final) 100^{th} term is 2%. Thus, in order to find the common difference (often denoted by d); we work it out as follows:

$1^{st}\text{ term} \Rightarrow \quad a = 0.15 \qquad --- \qquad 4.4(a)$

$100^{th}\text{ term} \Rightarrow \quad a + 99 \times d = 0.02 \qquad --- \qquad 4.4(b)$

putting 4.3(a) *into* 4.3(b) *yields*:

$$d = \frac{(0.02 - 0.15)}{99} = -0.001313$$

From the above, it can be inferred that, the desired sequence of target returns starting from the initial target supplied as input would be: **0.150000, 0.148687, 0.147374, 0.146061 ...** and so forth up to the last (100^{th}) term value of **0.02000**.

4.5 Handling the return and budget constraints

This section discusses the handling of the basic (practical) constraints in our unconstrained PSP formulation. Fuller detail on how we handled our constrained implementation will follow in the relevant sections.

4.5.1 Handling Return Constraint:

Before we delve into explaining how we handled our return constraint, we feel it is important to distinguish between the objective and cost functions, as both are often inter-switched. The former, mostly represents the function that needs to be maximized/minimized in solving the optimization problem; while the latter often represents the function tasked with guiding the search process towards promising regions of the search space. Although, nothing hinders an objective function from serving as a cost function; however, di Tollo and Roli [34] posit that, search processes have more chance of being guided towards promising solutions when using a cost rather than an objective function.

In view of the above and going by the fact that, in our unconstrained PSP formulation; one of the only two constraints that are likely to be violated as the search progresses, is the return constraint; we decided to use a cost rather than objective function to solve our unconstrained bi-objective optimization problem. We designed our cost function in such a way that, it will allow us to penalize any violation of the return constraint,

this enables the algorithms to make a rigorous search in order to make sure that the desired target is achieved, while at the same time minimizing the risk. Our cost function incorporates a penalty term, $\lambda \in [0, 1]$ – a cost value associated with penalizing return constraint's violation, helps in achieving a trade-off in minimizing portfolio risk and bridging the gap between the portfolio's return and the desired target return.

The cost function is a weighted sum of the two components (portfolio risk and return); and takes the form:

$$Min\ [\ (1-\lambda)\,Portfolio\ Risk + \lambda \left|\ Portfolio\ Return - Target\ Return\ \right|\ ] \quad - \quad 4.5.1$$

From equation *4.5.1* above, it can easily be inferred that, the penalty value λ plays an important role in achieving a trade-off in minimizing the violation of return constraint and the magnitude of portfolio risk.

To understand the effect of λ, let's now consider the two extreme values it can assume. Suppose an λ takes a maximum value of 1, this will mean that, there is a very strong likelihood that the algorithm will find a portfolio configuration whose portfolio return will (almost) exactly match the desired target, irrespective of the magnitude of its risk. On the other hand, if λ were to assume a value of zero, the resultant portfolio configuration would be one in which minimizing the portfolio risk takes utmost priority over achieving the given target.

After running our algorithms several times with different values of λ, we found that an λ value within the interval [0.65, 0.7] has been found to work satisfactorily well with most of our algorithms' implementations, as it is found that λ values within this

interval ensure that solutions obtained are still able to reach the target but at a reasonably moderate risk value.

In order to save time and ensure that all solutions returned by our algorithms are reasonably good, we modified our algorithms in such a way that whenever a final solution to a given target is found, such solution is then passed as a starting solution for the next target.

4.5.2 Handling Budget Constraint:

As it is believed that a repair approach for handling constraints' violation provides a trade-off between diversification and intensification [34]; we decided to implement such an approach in satisfying the budget constraint of our unconstrained PSP formulation.

In order to ensure that, the budget constraint is satisfied, after each iteration we decided to normalize the weights so as to sum up to unity. This approach repairs assets weights in the following way:

$$w_i^* = \frac{w_i}{\sum_i w_i}; \quad \forall i \qquad --- 4.5.2$$

Where w_i^* stands for the repaired (normalized) weight of asset i, while w_i represents actual (unrepaired/un-normalized) weight of asset i. Notice that $\sum_i w_i \neq 1$, while $\sum_i w_i^* = 1$.

4.6 Performance Metrics & Evaluation of algorithms

In this section we present some metrics proposed in the literature for evaluating the performance of our algorithms in solving multiobjective optimization problems. With these performance measures, we will be able to evaluate the success of our algorithms in solving a bi-objective PSP; we will also use these metrics in order to compare our algorithms against each other. For both datasets, CPLEX solver and our algorithms solved the same instance of PSP supplying (as part of the input set) 200 equally-spaced target returns, bringing the total number of points/portfolios on the various efficient frontiers generated by our algorithms and the solver to 200. In order to achieve utmost numerical precision, all results were rounded to 9 decimal places.

In order to evaluate the performance of multiobjective optimization techniques, there is the need to invoke the help of some performance measures. In doing so, it is important to have more than one metric in evaluating such performances. Zitzler [156] suggested for computing at least N performance metrics for an *N-objective* optimization problem. Deb [32] suggested for the classification of performance metrics into three different classes: those for convergence, diversity and metrics for both.

According to Jaszkiewicz [83], evaluation of algorithmic performance in solving multiobjective optimization problems should consider two main measurement criteria; and these comprise of: (1) *computational* requirement and, (2) the *quality* of the returned solutions. The *quality* metrics can be further subdivided into either *cardinal* or *geometrical*; where the cardinal measures (which quite often use relations such as equivalence and/or dominance) enumerate some number of points/solutions satisfying

some conditions. On the other hand, the geometrical measures consider the geometrical position of the nondominated solutions in the objective space.

According to Zitzler *et al.* [157], there are three main goals in Pareto multiobjective search that need to be identified and measured, including:

(a) Convergence: The convergence metrics mostly measure minimum distance of the obtained nondominated solution to the actual Pareto front (if it is known).

(b) Non-uniformity of Pareto front: Non uniformity metrics evaluate how good is the distribution of the obtained solutions, and

(c) Coverage: These metrics aim at maximizing the size of the obtained nondominated front (i.e. for each objective, a wide range of values should be covered by the nondominated solutions).

In all cases, the set of exact solutions produced by the quadratic optimization software (CPLEX 11.2) forming a frontier of optimal portfolios is regarded as the **true Pareto frontier** which serves as a benchmark (reference) solution upon which all other approximate Pareto solutions generated by our algorithms are evaluated.

In view of the suggestion made above by Zitzler *et al.* [157], despite the fact that our PSP is a bi-objective optimization problem, we are going to evaluate our algorithms based on the three metrics discussed below:

4.6.1 Convergence Metric:

In order to measure the degree of convergence and how accurate our algorithms were able to estimate the UEF, we decided to apply one of the solution quality metrics used in Cura [26]. This performance measure is referred to as the Mean Euclidean Distance, *mEd,* which is used to measure the area between the *optimal UEF* generated by the CPLEX solver and the one generated by an algorithm. We can now define the *mEd* as follows:

Let the pair $\left(v_i^{CPLEX}, \mu_i^{CPLEX}\right)(i = 1,...,\psi)$ be the variance and mean return of a point/portfolio i on the solver's efficient frontier; and let the pair $\left(v_i^{A}, \mu_i^{A}\right)(i = 1,...,\psi)$ be the variance and mean return of a point/portfolio i on the efficient frontier produced by algorithm A. Let also $R_i^T\ (i = 1,...,\psi)$ be the target return to be achieved at point i, where in all cases $\psi = 200$. Thus,

$$mEd = \left(\sum_{i=1}^{\psi}\sqrt{\left(v_i^{CPLEX} - v_i^{A}\right)^2 + \left(\mu_i^{CPLEX} - \mu_i^{A}\right)^2}\right) \times \frac{1}{\psi} \qquad - \qquad 4.6.1$$

A $mEd = 0$ means that algorithm A is able to perfectly produce the true Pareto front of the optimization problem at hand; i.e. algorithm A is as effective (in generating the optimal UEF) as the nonlinear solver (CPLEX 11.2). Thus, higher value of *mEd* reveals the degree of algorithm A's ineffectiveness; hence, lower values of *mEd* are always desired.

4.6.2 Coverage Metric:

The main idea behind measuring this metric introduced by Zitzler *et al.* [157] is to compare two different Pareto optimal solutions against each other by considering a dominance relation.

Now with respect to our research, which is a bi-objective PSP; we give much emphasis on two important terms by minimizing the portfolio risk while at the same time bridging the gap between the portfolio return and the desired target. Now in order to define what a *dominance relation* ($\succ$) means with respect to our type of problem, we incorporated these two terms (return and risk) in determining when a given solution $r \in R$ is said to dominate/outperform a corresponding solution $s \in S$.

Let the pair $\left(v_i^R, \mu_i^R\right)(i = 1,...,\psi)$ be the variance and mean return of a point/portfolio i on the approximate (Pareto) efficient frontier generated by algorithm R; and let the pair $\left(v_i^S, \mu_i^S\right)(i = 1,...,\psi)$ be the variance and mean return of a point/portfolio i on the efficient frontier produced by algorithm S. Let also $R_i^T\ (i = 1,...,\psi)$ be the target return desired to be achieved by portfolio i. We can now define a strong dominance relation by determining:

$$IF \begin{cases} \left(\left|\mu_i^R - R_i^T\right| < \left|\mu_i^S - R_i^T\right|\right) \& \left(v_i^R < v_i^S\right) \\ or\ \left(\left|\mu_i^R - R_i^T\right| = \left|\mu_i^S - R_i^T\right|\right) \& \left(v_i^R < v_i^S\right) \\ or\ \left(\left|\mu_i^R - R_i^T\right| < \left|\mu_i^S - R_i^T\right|\right) \& \left(v_i^R = v_i^S\right) \end{cases} THEN\{R \succ\succ S \text{ at point } i\} \quad - \quad 4.6.2(a)$$

With the above relation, one can easily determine the total number or the ratio of points in which solutions returned by algorithm R (*strongly*) dominates their counterparts produced by algorithm S.

There is, however, another way of defining a dominance relation with respect to our type of problem. Suppose we let $f_i(R) = (1-\lambda)v_i^R + \lambda\left|\mu_i^R - R_i^T\right|$ be the optimized cost function (as in equation 4.5.1) returned by algorithm R in achieving the i^{th} target return. Correspondingly, if we let $f_i(S) = (1-\lambda)v_i^S + \lambda\left|\mu_i^S - R_i^T\right|$ be the optimized cost function returned by algorithm S in achieving the same i^{th} target. We can define a *weak* dominance as follows:

$$IF\left\{f_i(R) < f_i(S)\right\} THEN\left\{R \succ S \text{ at point } i\right\} \quad - \quad 4.6.2(b)$$

When the above relation is satisfied, we can say that algorithm R *weakly* dominates algorithm S at point i. However, in our analysis we decided to conduct our algorithmic analyses with the *strong* dominance relationship as described in equation 4.6.2(a) above.

Now, let $C(R, S)$ be the coverage metric when two Pareto front sets (R and S) are compared; mapping the ordered pair (R, S) to the interval [0, 1]. It can be measured by:

$$C(R, S) = \frac{\left|\left\{s \in S \mid \exists r \in R : r \succ\succ s\right\}\right|}{|S|} \quad - \quad 4.6.2(c)$$

Where $|S|$ stands for the total number of solutions in set S and $\succ$ signifies a dominance relation. Thus, $r \succ\succ s$ simply means solution $r \in R$ *strongly* dominates $s \in S$ (i.e. the objective values of r are better than those of s). In a nutshell, $C(R, S)$ shows the proportion of the number of solutions in S that are dominated by the solutions in R. A $C(R, S) = 1$ means all solutions in S are dominated by corresponding

solutions in R; while $C(R, S) = 0$ implies no solution in S is dominated by a corresponding solution in R. Furthermore, if relation $C(R,S) > C(S,R)$ holds, it simply means that set R has recorded more better solutions than S did. It should also be noted that, $C(R, S)$ is not necessarily equal to $1 - C(S,R)$. It is therefore obvious that, there are situations when the C-metric cannot decide if a given solution dominates the other and vice versa.

4.6.3 Non-uniformity of Pareto front:

In order to measure the degree of uniformity or otherwise inherent in the distribution of a given Pareto front; Lee *et al.* [95] proposed a $D(\bullet)$ metric. Suppose R is a set of Pareto front; the quantity $D(R)$, which measures the distribution of the Euclidean distance (d_i) between two consecutive solutions along the Pareto front is given by:

$$D(R) = \sqrt{\frac{\sum_i \left(d_i/\bar{d} - 1\right)^2}{|R| - 1}} \qquad - \qquad 4.6.3(a)$$

The numerical value of $D(\bullet)$, quantifies the standard deviation of the distances (d_i) normalized by the average distance, $\bar{d}$. A $D(R) = 0$ implies that there is a uniform spacing in the R's Pareto front. Thus, higher values of $D(R)$ signifies non-uniformity in the spacing of the R's Pareto front. Therefore, going by what has been described above, a lower value of $D(R)$ is desired. It should be understood that the $D(\bullet)$ metric can only be suitable and meaningful for a bi-objective optimization problem, due to the fact that, it is very unclear how "consecutive" can defined in a more than two objective problem.

As per our PSP problem, by defining the pair $\left(v_i^R, \mu_i^R\right)(i = 1,...,\psi)$, as described in section 4.6.2 above; we can compute a Euclidean distance (d_i) between two consecutive points (i & i+1) along the Pareto frontier by:

$$d_i = \sqrt{\left(v_i - v_{i+1}\right)^2 + \left(\mu_i - \mu_{i+1}\right)^2} \qquad - \qquad 4.6.3(b)$$

4.7 Results & Discussions

In this section we present the results obtained by running a simulation of our designed algorithms and separately a nonlinear optimization solver (CPLEX 11.2); purposely for comparing their performance against each other by using some standard performance evaluation metrics used by other researchers in the optimization community. For both sets of data (sourced from *Hang Seng* and *FTSE100* indices); the CPLEX solver and our algorithms solved the same instance of PSP supplying (as part of the input set) 200 equally-spaced target returns, thus giving us 200 points/portfolios on the various efficient frontiers generated by our algorithms and the solver in which all numerical values were rounded to 9 decimal places. The results were obtained after taking the average of the outputs of 35 experimental trials for each algorithm (with its best parameter configuration as outlined in section 4.3) to solve our unconstrained PSP formulation.

Now in order to obtain a great numerical precision from our solver-generated exact (optimal) solution, we decided to change the solver directives settings that are responsible for producing very high numerical precision results from the default settings. These perturbed directive settings are:

(a) ***qcpconvergetol*** (default = $\mathbf{10^{-7}}$)

Inserting the above directive in our AMPL script allows for setting the convergence tolerance on complimentarity in quadratically constrained problems (such as the constrained case of our PSP formulation). The barrier algorithm (the solver uses) terminates with an optimal solution if the relative complimentarity is smaller than this value. The default value of this tolerance limit (as can be seen above) is just $\mathbf{10^{-7}}$. However, in our implementation we decided to use an extremely smaller value of 10^{-10} as doing so results in greater numerical precision.

(b) ***comptol*** (default = $\mathbf{10^{-8}}$)

The other directive we used in our AMPL script is the ***comptol*** directive which can be used to obtain higher numerical precisions for both linear (LPs) and quadratic programming (QPs) problems when all the accompanying constraints are linear. As in (a) above, the barrier algorithm returns an optimal solution if the relative complimentarity is smaller than this value. The default value of this tolerance limit (as can be seen above) is just $\mathbf{10^{-8}}$. However, in our implementation we decided to *raise the bar higher* by using even an extremely smaller value of $\mathbf{10^{-10}}$; as doing so results in greater numerical precision, which will definitely help in testing the ability of our metaheuristic techniques.

In order to critically analyze and evaluate the performance of our designed algorithms, we plan to present and discuss the results obtained by explaining the outputs in respect of the three performance and evaluation metrics (convergence, coverage and uniformity) as discussed in section 4.6 above coupled with other pictorial/graphical representations aimed at showing the effort put on by the algorithms.

4.7.1 Algorithmic analysis based on convergence ability:

As already stated before, this research considers to be a bi-objective PSP, we give much emphasis on minimizing the portfolio risk while at the same time bridging the gap between the portfolio's and the supplied target returns; which is why our convergence evaluation measure (the *mEd*) incorporates these two terms. An important graphical tool to showcase algorithmic performances in solving an unconstrained PSP in this research can be depicted by plotting the various portfolio risks against their corresponding portfolio returns on a risk-return plane. The result is a parabola-like plot of points (portfolios) seemingly forming a frontier of nondominated points collectively known as *efficient frontier (EF)* in financial literature. Each of them represents a combination of several proportions of investment's funds allotted to some carefully chosen stocks offering a trade-off between maximization of portfolio return and minimization of portfolio's risk of investment. Typical example of EFs generated by a nonlinear optimization solver (CPLEX 11.2) and our algorithms for the two different datasets used in this research can be seen in figures 13(a) and 13(b) below:

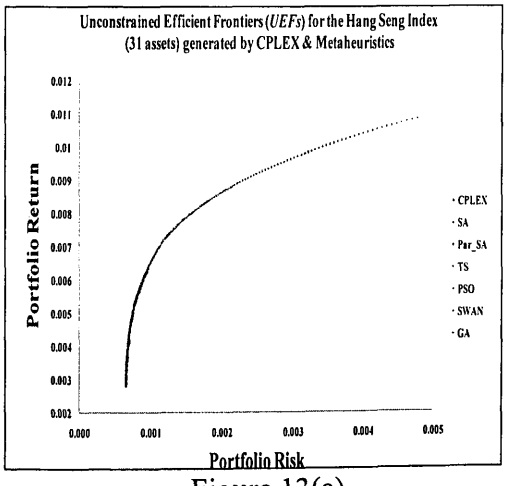


Figure 13(a)

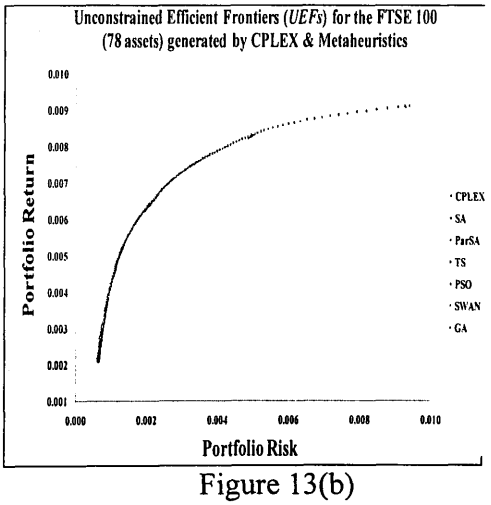


Figure 13(b)

Figure 13: showing *mEds* obtained by all the algorithms.

From the results obtained, it is extremely difficult to tell the difference existing among the various Pareto fronts shown in both figures 13(a) and 13(b); as mere eyeballing the figures suggest that, all the 6 designed algorithms were able to produce outputs (*UEFs*) that seem to be intimately comparable to those returned by the solver; and even this achievement says much on their performances in respect of solutions' quality. CPLEX solves these problems in about two seconds.

The following table summarizes the numerical results of the convergence performance metric obtained by our algorithms for the two datasets used in this research.

Index	Assets	Metaheuristic Algorithms (*h*)	Convergence Metric *Mean Euclidean Distance* *mEd* ($\times 10^{-6}$)	Average Execution Time per solution (secs)
Hang Seng	31	SA	6.4998	1.19
		ParSA	4.6334	1.79
		TS	9.0278	1.08
		PSO	0.1215	2.25
		SWAN	0.0056	2.55
		GA	4.8751	2.12
FTSE 100	78	SA	12.431	9.19
		ParSA	5.8621	6.29
		TS	22.157	6.97
		PSO	1.6994	6.03
		SWAN	1.2818	6.65
		GA	9.9486	7.93

Table 1: Showing values of the *mEd* and average execution time

From Table 1 above, it can easily be observed that the numerical values of the *mEds* are (in all cases across the various algorithms) very close to zero indicating that virtually no significant difference exists between the results generated by the solver and our algorithms in solving the unconstrained case. Recall that from section 4.6.1 above, an *mEd* value equal to or very close to zero is much desired as it indicates the strength of a given algorithm's ability in estimating the true Pareto front. It can also be noticed that, although with less average execution time, the TS recorded a worst performance with respect to an *mEd* value, (having maximum values of 9.0278 and 22.157 respectively for both *Hang Seng* and *FTSE 100* indices) when compared to any other of the remaining five algorithms. On the other extreme end, however; our newly

designed algorithm – SWAN recorded the best performance in both datasets with 0.0056 and 1.2818 *mEd* values respectively. Even though, it recorded the highest average execution time, we still believe an average time of at most 2.6 secs is still reasonable enough for such a remarkable performance. The PSO seems to be the next best algorithm after SWAN, even though it also recorded a high but reasonably acceptable average execution time when compared to other algorithms.

In both cases, parallel SA seems to slightly outperform the GA. For the smaller Hang Seng dataset, the parallel SA has an *mEd* value of 4.6334 as against the GA's 4.8751; similarly, parallel SA recorded an *mEd* value of 5.8621 in the FTSE100 index as against 9.9486 for the GA. SA, unlike TS, provided a much better solution than the latter, although (as expected) at the expense of larger execution time; its *mEd* value in both datasets is much smaller than those credited to TS, indicating a stronger ability in estimating the true Pareto front. Of particular interest is the GA, which to our dismay performed badly as against our expectation of its competitive potential, that we thought would match that of our newly designed hybrid method and especially PSO, being in the same category as highly revered EAs. In order to further visualize how our algorithms performed against each other, we decided to depict the positions of their *mEd* values on a radar plot in figures 4.7.1(c) and 4.7.1(d) below:

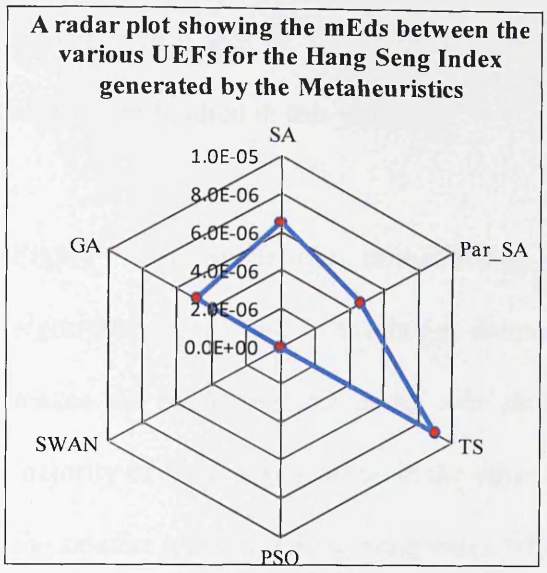


Figure 14(a)

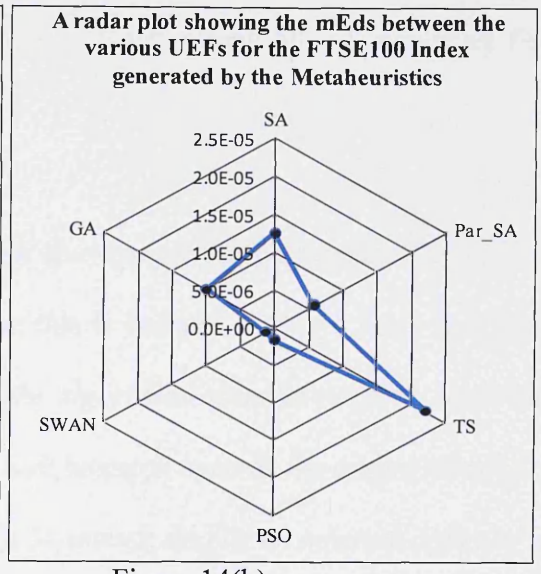


Figure 14(b)

Figure 14: showing mEds obtained by all algorithms for the Hang Seng and FTSE100 indices respectively

In each of the two figures [14(a) and 14(b)] above, it should be understood that, the closer a given point is to the origin (centre of the innermost hexagon), the better the corresponding algorithm's ability in estimating the true Pareto front.

Now considering figure 14(a) above, it will be extremely difficult to distinguish the performance of PSO from that of the SWAN; both of whom points seems to be 'spot-on' right at the origin; and mere eyeballing their respective points on the radar plot reveals the degree at which both algorithms performed in relation to other algorithms. The duo, by estimating the Pareto front with a very high degree of precision, further attest to the power of EAs in solving difficult optimization problems. However, GA (another member of EAs family) which is left trailing behind seems to be fiercely competing for the 3rd position against the parallel SA, both of whose points fall in the third inner hexagon. As can be seen from the same figure; although, SA positioned itself in the 2nd outer hexagon being far away from the origin, its performance seems to

be better than that of TS which is found to be worst among all the remaining five algorithms studied in this instance.

Figure 14(b), we believe, is misleading in showcasing the performance of the six algorithms in relation to the larger dataset; this is because, taking a hasty look at it makes the reader wonder as to: *why do the algorithms seem to perform better* (as majority of them appear to be in the innermost hexagon towards the origin) *relative to the smaller problem (*Hang Seng index with 31 assets*)*; *despite an inherent difficulty of the search process due to, especially a remarkable increase in the problem size/dimension* (recall FTSE100 index has 78 assets). In view of this (as the reader might expect), the reverse is the case (i.e. performance of the algorithms in the smaller [Hang Seng 31 assets] dataset is indeed better across all the algorithms), as it is easier to handle a comparatively smaller search space. The proof to the above statement can only be plainly noticed when performance values from both datasets are plotted on the same radar plot (having the same axes) as can be seen on figure 15 below:

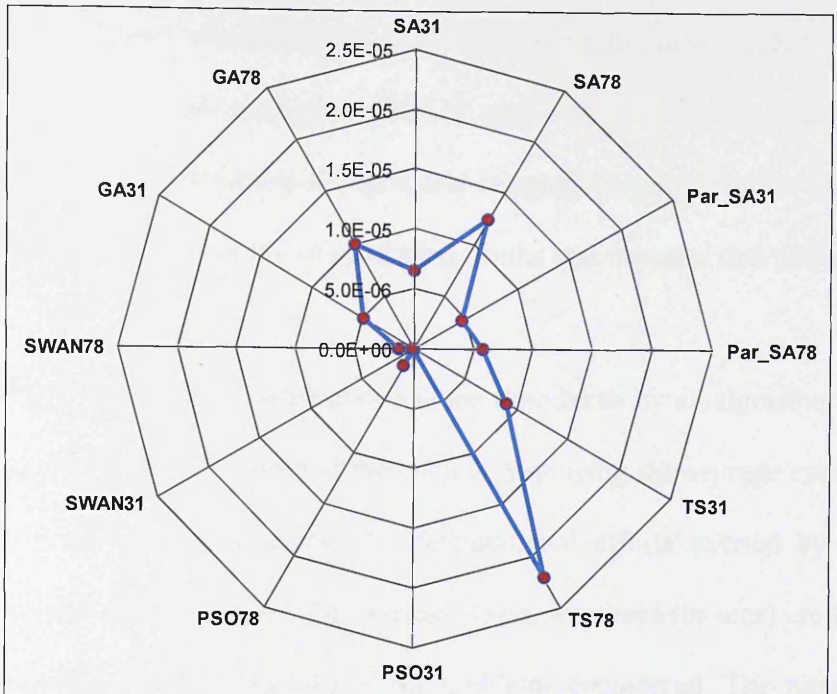


Figure 15: showing mEds of all the algorithms for the Hang Seng and FTSE100 Indices plotted on the same axis.

To make it clearer for the reader as to which of the points on Figure 15 above is for which algorithm and/or for which of the datasets; we add a numerical suffix equivalent to the problem size to each of the abbreviations for the six algorithms. For instance, a point depicting an *mEd* value on "GA78" axis shows the magnitude of GA's performance on the larger dataset (FTSE100 index); while a point on "SA31" axis of the plot signifies the performance of SA on the smaller dataset (Hang Seng index). Furthermore, for easier comparison we place axes of the same algorithm next to each other. For instance, it can be seen that "SA78" is next to "SA31", "Par_SA78" is next to "Par_SA31", and so forth.

Therefore, it can be noticed from the above Figure 15 that, there is a decline in performance across all the algorithms; however, in some the decline is more apparent and easily more noticeable than in others. For example, the decline is more visible in TS, followed by SA, GA and then Par_SA. As for the PSO and SWAN their

performance has been very consistent and encouraging; in the sense that, the problem size seems to have a very negligible effect (if any) on the duo. Both seem to maintain their positions in the innermost hexagon and clinging very tight to the origin, which simply translates to their ability of producing results that matched that of the solver.

Another indicator of performance ability is the time taken by an algorithm to return a solution on the average. Looking at the column displaying the average execution time on Table 1 will reveal the degree of computational efforts exerted by the search methods in providing a solution. The average execution times (in secs) are found to be very reasonable, going by the size of the problems considered. The times recorded would be appreciated and considered acceptable, given the fact that even for the smaller 31 asset problem, a total of 992 data items (including a square covariance matrix with 961 data items and a vector of 31 expected returns) are required as inputs. Similarly, in solving the larger 78 assets problem apart from the vector of 78 expected returns, there is also a square covariance matrix consisting of 6084 data items that are necessarily required as inputs. It is quite interesting to see that, despite having such dense matrices as inputs, our algorithms are able to find a solution in as low as 1.08*secs* for TS in the 31 asset problem, although at the expense of convergence. However, other methods especially PSO and SWAN are able to return quality solutions in a quite reasonable time frame without any compromise on performance.

Furthermore, our algorithms proved to be robust since the parameters we used in running our algorithms were not further fine-tuned in order to solve even the larger problem. The above assertion can be easily supported by observing their performances when dealing with an even larger dataset from the *FTSE 100* Index.

4.7.2 Algorithmic analysis based on coverage ability:

After conducting some preliminary simulation runs, we observed that in extremely rare cases the solutions returned by our algorithms do slightly *dominate* those of the solver's; as our algorithms were able to meet the target at lower risk as defined in equation 4.6.2(a) above. This is because the settings of the solver are such that they do not guarantee an exact optimum solution. However, this dominance is extremely negligible to the tune of 10^{-9} on the average. It is in view of this, we decided to include the solver's outputs in Tables 2 and 3 in order to show the ratios in which our algorithms were able to outperform the solver and vice versa.

The *C*-metric is an essential quantitative tool used in evaluating algorithmic performance as discussed in section 4.6 above. We are now going to use it to discuss the performances of the algorithms against the nonlinear solver as well as against each other for the two different datasets used in this study. Primarily, we will discuss a dominance relation especially where a given algorithm is found to outperform others.

The following table gives the result of dominance relation existing between the various algorithms obtained by solving the smaller 31 assets (Hang Seng index) PSP.

C(*Algorithm1*, *Algorithm2*)		*Algorithm2*						
		CPLEX	SA	Par_SA	TS	PSO	SWAN	GA
Algorithm1	CPLEX	–	0.920	0.805	0.900	0.680	0.665	0.975
	SA	0.005	–	0.140	0.505	0.015	0.015	0.345
	Par_SA	0.005	0.430	–	0.710	0.025	0.020	0.495
	TS	0.005	0.090	0.030	–	0.020	0.020	0.185
	PSO	0.010	0.915	0.780	0.890	–	0.295	0.950
	SWAN	0.015	0.920	0.795	0.895	0.355	–	0.975
	GA	0.000	0.295	0.290	0.185	0.000	0.000	–

Table 2: Showing the C-metric values for all algorithms against each other for the Hang Seng dataset

From Table 2 above, it can easily be observed that, the solver's solutions overwhelmingly dominate (see the row marked: CPLEX under *Algorithm1*) most of the solutions produced by our algorithms. However, there are mostly very rare cases in

which the solver's results were dominated also (see the column marked: CPLEX under *Algorithm2*). For instance, it can be seen that, the solutions produced by the duo of PSO and SWAN dominate the solver's by 1.0% (in 2 out of the 200 solutions) and 1.5% (only 3 out of 200) respectively. The trio of SA, Par_SA and TS managed to outperform the solver at only one solution (0.5%) each; while the GA was unable to dominate the solver at any point on the *EF*. A little bit down the table, it is interesting to see how the duo of PSO and SWAN perform very similarly with SWAN slightly having an *upper hand.* SWAN dominates SA, Par_SA, TS and GA by 92%, 79.5%, 89.5% and 97.5% respectively; while PSO follow suit with 91.5%, 78%, 89% and 95% for SA, Par_SA, TS and GA respectively. Furthermore, it can be inferred from the table, SWAN outperformed its closest competitor (PSO) in 71 (35.5%) out of the total 200 solutions generated; while on the opposite PSO dominated a total of 59 (29.5%) out of the 200 solutions generated by the SWAN.

The performance of GA under the dominance relation described in equation 4.6.2(a) against either the solver or any of the other 5 algorithms is surprisingly poor. This fact can easily be observed when the reader noticed that, GA is the only algorithm that was unable to dominate a single solution in the trio of CPLEX, PSO and SWAN; and even in situations where it managed to dominate other algorithms (SA, TS and Par_SA); the degree of such dominance is not up to 30% (the maximum being 29.5% against the SA). Observing GA's performance from another angle also reveals its outright failure. For instance, taking a quick look at the column marked: GA under *Algorithm2*, the reader can easily notice that GA has the most number of solutions dominated by the duo of CPLEX and SWAN up to the tune of 97.5% (195 out of the entire 200 generated).

Now, if we consider the performance of SA against TS, we realize that CPLEX dominated SA by 92% as against the 90% of TS. Similarly, PSO outperformed 91.5% of the solutions generated by SA as against the 89% of the TS; while SWAN to SA is 92% as against 89.5% for the TS. In the same vein, GA dominated SA by 29.5% as against only 18.5% for the TS; therefore, with these results one can easily be misled to believe that TS is better than SA. However, we argue that comparing the performances of SA and TS based on Table 2 only will give a contradictory conclusion as can be inferred on Table 1, where it is made apparently clear that SA recorded lower *mEd* values (in both datasets) when compared to TS; and hence adjudged better. It should be understood that, the convergence metric (*mEd*) described in equation 4.6.1 takes into account all the 200 solutions generated on the *EF*; while the coverage metric resulting from equation 4.6.2(a) considers only solutions that satisfy the conditions set in the *strong* dominance relation described in equation 4.6.2(c). By this we are still of the view that, as far as the smaller dataset problem is concerned, SA performed better than TS.

The next Table 3 shows the output obtained by computing the C-metric values for the solver and the designed algorithms in solving the larger 78 asset dataset problem.

C(Algorithm1, Algorithm2)		*Algorithm2*						
		CPLEX	SA	Par_SA	TS	PSO	SWAN	GA
Algorithm1	CPLEX	–	0.815	0.775	0.760	0.250	0.200	0.865
	SA	0.010	–	0.025	0.390	0.010	0.010	0.315
	Par_SA	0.060	0.785	–	0.860	0.125	0.105	0.595
	TS	0.035	0.180	0.105	–	0.010	0.005	0.325
	PSO	0.115	0.870	0.690	0.845	–	0.065	0.720
	SWAN	0.125	0.895	0.385	0.865	0.130	–	0.810
	GA	0.005	0.210	0.170	0.265	0.020	0.015	–

Table 3: Showing the C-metric values for all algorithms against each other for the FTSE100 dataset

Table 3 above reveals a decline in the solver's and other algorithms' performances in comparison with what is obtainable on Table 2, as even the solver seems to be feeling the impact of dealing with a larger dataset of 78 assets (which translates to approximately 150% increase in size) when compared to the smaller 31 assets dataset analysis presented on Table 2.

A quick look at the row marked CPLEX under *Algorithm1*, would definitely show a sudden drop in the solver's performance when compared to the corresponding row in Table 2. On Table 3 above, CPLEX dominated the solutions generated by SA, Par_SA, TS, PSO, SWAN and GA by 81.5%, 77.5%, 76%, 25%, 20% and 86.5% respectively as opposed to the corresponding values of 92%, 80.5%, 90%, 68%, 66.5% and 97.5% on Table 2. Furthermore, a mere glance at the column tagged CPLEX under *Algorithm2*, shows that, SA now dominated 2 solutions (1%) produced by CPLEX as opposed to only 1 (0.5%) solution in the smaller dataset problem. Par_SA dominates 6% as opposed to 0.5%, TS - 3.5% as opposed to 0.5%, PSO – 11.5% as opposed to the previous 1%, SWAN – 12.5% as opposed to the previous 1.5%, and GA has now managed to dominate only 1 out of the entire 200 solutions generated by the solver as opposed to none in the smaller problem.

Despite the sudden drop in performance, the duo of PSO and SWAN maintained their tight competition with SWAN leading the way again. In all other algorithms, except for the Par_SA, it is easily noticeable that SWAN dominates more solutions than its closest competitor – the PSO. The GA on the other hand recorded a slight improvement in terms of the number of algorithms dominated; as in this case it managed to dominate only 1 solution (0.5%) returned by CPLEX, 4 solutions (2%)

from PSO and 3 (1.5%) from SWAN; unlike in the previous case where it was unable to dominate even a single solution returned by the trio of CPLEX, PSO and SWAN.

4.7.3 Algorithmic analysis based on uniformity of solutions:

The (uniformity) metric often denoted as $D(\bullet)$, as described in section 4.6.2 is aimed at identifying the uniformity of the distributions of solutions inherent in the results generated by our heuristic methods.

$D(Algorithm)$ $(\times 10^{-5})$			
Index		Hang Seng	FTSE100
Assets		31	78
Algorithm	SA	5.1965	8.5266
	Par-SA	5.0065	8.2996
	TS	5.0441	8.4995
	PSO	5.0339	8.2998
	SWAN	5.0338	8.4939
	GA	5.0329	8.4942

Table 4: Showing the values of uniformity metric for all algorithms from the two datasets used

From Table 4 above, it clear that the results generated by our algorithms are uniformly distributed, as the $D(\bullet)$ values obtained from our algorithms are very close to zero (i.e. all to the tune of 10^{-5}). It is also evident from the table that, the problem dimension has very little effect (if any) at how any of the algorithms generates its solutions along the *EF*.

Although in both datasets considered, all the results favored Par_SA as the algorithm with highest degree of uniformity (recall that a lower value of $D(\bullet)$ is desired) with 5.0065×10^{-5} and 8.2996×10^{-5} for *Hang Seng* and *FTSE100* indices respectively; however, it is evident that the edge it has over other algorithms is very negligible, and it seems all the algorithms have the same degree in generating uniform results.

Section 4.7.3 has nothing to do with an algorithm's degree of convergence or coverage, but it is rather concerned with whether results generated are uniformly distributed or not. Thus, now based on the results presented in sections 4.7.1 and 4.7.2 above, SWAN has the most ability to estimate the true Pareto front, hence, we can now conclude it is the best performing algorithm and closely followed by PSO.

5.0 Constrained PSP

5.1 The Constrained case

The main aim of this chapter is to introduce and describe in detail our newly developed neighbourhood structure for solving a constrained version of the PSP studied in this research. The constrained formulation is a special case of PSP in which the final solution of the optimization problem has to fulfill some set of conditions. However, imposing these conditions has some accompanying consequences, among which is the risk of the problem becoming intractable as well as susceptible to falling within some class of difficult optimization problems regarded as NP-hard; thereby making it much harder or even (in most cases) impossible to solve by the conventional exact methods embedded in most of the state-of-the-art nonlinear optimization solvers. There are, however, quite number of practical constraints that are often incorporated into the constrained PSP; a brief description of some of them can be found in section 2.4; but as far this research, we plan to incorporate only two among the practically implementable constraints – namely the cardinality and floor & ceiling constraints.

Cardinality constraint: To implement this constraint in our formulation, we decided to limit the number of assets that the portfolio composes. A value of $k \leq n$ (typically 5 or 10) is chosen, such that the number of assets selected to constitute a portfolio is no more than k.

Floor & Ceiling constraint: For this constraint, the weight of each asset selected to form part of a portfolio is limited and lies within an interval, in which the *minimum*, l_i and *maximum*, μ_i weights for each asset i are given. In our formulation, we impose

that if an asset i is selected to be part of a given portfolio its weight w_i must satisfy $l_i \leq w_i \leq \mu_i$, otherwise it is set to zero.

For easier handling of the above constraints, we adopt the method used in Chang *et al* [20] by declaring a binary variable δ_i for each asset i in the universe of assets, taking a value of 1 if asset i is included in the portfolio and a value of 0 otherwise. With this, we now come up with a constraint pair:

$$\sum_{i=1}^{n} \delta_i \leq k \quad \Rightarrow \quad \textit{cardinality constraint}$$

$$\delta_i l_i \leq w_i \leq \delta_i \mu_i \quad \Rightarrow \quad \textit{floor \& ceiling constraint}$$

The basic PSP, with the incorporation of the constraint pair above, becomes a mixed-integer quadratic programming problem whose solution is much more computationally difficult to find using the conventional methods.

Recall that, in this research, we designed and implemented six different metaheuristic algorithms to tackle the unconstrained case of the PSP; out of which three (SA, TS and Par_SA) are Local Searches, and the remaining three (GA, PSO and SWAN) are Evolutionary Algorithms. Furthermore, recall that, GA's performance in the unconstrained case of the PSP (previous chapter) has not been so encouraging. So, going by the poor performance recorded by GA in the less difficult (unconstrained) case as presented in section 4.7 above; we decided not to implement a constrained version of the PSP using GA.

The difficulty inherent in finding a very good solution within a practically reasonable time frame in the constrained formulation of the PSP warrants the need to assist and

guide the algorithms towards promising regions of the search space; and this can be achieved by developing a sound and effective neighbourhood structure for this purpose. Major contributions that our research has to offer to the academic knowledge in general and optimization communities in particular are the development of two new neighbourhood structures that are entirely different from (despite being inspired by) those presented in the work of Schaerf [129]. In this chapter we are going to describe how we implemented these two versions of neighbourhood structures as will be described in sections 5.3 and 5.4 below. To the best of our knowledge, the way we implemented these neighbourhood generating mechanisms is unique, and have never been reported in any literature before. The first one, entitled: IDDIT (based on the processes involved therein) is aimed at guiding the Local Searches (SA, TS and Par_SA) in generating a neighbouring solution from the incumbent, while at the same time ensuring that all constraints are satisfied.

The other neighbourhood mechanism is designed in such a way that it will be much more suitable for our designed swarm algorithms (PSO and SWAN). It comprises processes similar to those found in IDDIT, aimed at generating neighbouring solutions from a swarm of particles. While ensuring feasibility of solutions and satisfiability of constraints, this neighbouring-solution generating mechanism, maintains some degree of interaction between a given particle (candidate solution) with its local best and global best solutions as can be obtained in the conventional implementation of the basic PSO and SWAN aimed at solving the unconstrained case. With the above brief explanation of how our neighbouring-solution generating mechanisms operate; we feel it is important to briefly describe the concept used in defining a candidate solution's representation in our constrained formulation of the PSP; and this is provided in the

following section. We provide a thorough explanation of each of the two neighbourhood relations in sections 5.3 and 5.4 below.

5.2 Solution Representation

Before we embark on thorough explanation about our neighbourhood structures, it is highly important to describe how a candidate solution (portfolio) is represented in our constrained case algorithmic implementations. It should be noted that, one of the crucial aspects of our constrained PSP implementation has to do with the efficient representation of a candidate solution. In this regard, we adopt the proposal provided by Chang *et al.* [20] in which a given portfolio is characterized by two main parts. The first denoted by L, is an integer set containing the indices of constituent assets in a given candidate solution (portfolio); while the other denoted by W, is a set of real numbers signifying the actual proportion of portfolio funds invested in corresponding assets whose indices make up set L.

For instance, suppose there is a universe, U of n assets; each asset $j \in U$ has a corresponding randomly generated real number $\left(i.e.\ w_j \mid 0 \le w_j \le 1\right) \forall j \in U$ which potentially (after some renormalization process is executed) becomes its actual proportion in a given portfolio fund. Now in order to represent a solution, we partition a candidate portfolio into 2 distinct parts; a set L containing *at most* k chosen assets and a set $\boldsymbol{W}$ of randomly generated real numbers $\left(w_j \mid 0 \le w_j \le 1\right)\ \forall j \in L$. All assets $j \in U$ have an associated binary variable, z_i taking a value 1 if asset j is included in portfolio $L\left(i.e.\ z_j = 1;\ \forall j \in L\right)$, otherwise it takes the value zero $\left(i.e.\ z_j = 0;\ \forall j \in \{U - L\}\right)$. All asset weights in the portfolio L $\left(i.e.\ w_j \mid \forall j \in L\right)$ must be

renormalized in order to serve as actual proportions of portfolio funds summing up to unity; while all other assets $j \in \{U - L\}$ will have their w_j values implicitly set to zero as far as portfolio L is concerned; because a given portfolio configuration involves multiplying asset weights multiplied by their corresponding binary variable $\left(i.e.\ w_j = w_j \times z_i \mid \forall j \in L\right)$.

Suppose we have a portfolio L containing k assets; so, in order to ensure that all assets in L at least satisfy the minimum threshold limit; we first allocate to each of them a weight equivalent to the minimum threshold limit ε_j, and this implies a fraction $\left(\sum_{j \in L} \varepsilon_j\right)$ of the total portfolio is already accounted for. Furthermore, in order to make sure that the actual weights of all assets in L now sum up to unity, each w_j can now be interpreted as relating to the share of the *free* portfolio proportion $\left(1 - \sum_{j \in L} \varepsilon_j\right)$ associated with assets $j \in L$.

To further explain our approach; suppose we have $n = 20$ as the total number that makes up the universe of assets, U, out of which no more than $k = 3$ assets should be included in any portfolio and each of the portfolio's constituent assets should have no less than $\varepsilon_j = 15\%$; one possible solution might therefore be: $L = \{3, 4, 8\}$ *and* $W = \{w_3 = 0.7,\ w_4 = 0.9,\ w_8 = 0.6\}$; this means our portfolio consists of assets 3, 4 and 8; and since we know that each of the three chosen assets must have a minimum proportion of 15%, then already $\sum_{j \in L} \varepsilon_j = 45\%$ has been accounted for, then the remaining *free* portfolio proportion $\left(1 - \sum_{j \in L} \varepsilon_j\right) = 55\%$. So the share of asset 3 in the *free* portfolio proportion will now be

$w_3/(w_3+w_4+w_8) = 0.7/2.2 = 0.3182$ and therefore, the *actual* portfolio proportion w_3 devoted to asset 3 can be obtained by adding the minimum proportion to asset 3's appropriate share of the *free* portfolio proportion will be $w_3 = 0.15 + 0.3182 \times 55\% = 0.325$. Similarly, $w_4 = 0.15 + 0.4091 \times 55\% = 0.375$ and $w_8 = 0.15 + 0.2727 \times 55\% = 0.300$.

It is important to note that, the main advantage of this approach is to ensure that all the w_j values satisfy the minimum proportion limit ε_j, while at the same time satisfying the budget constraint since they all sum up to unity. However, this approach does not guarantee satisfying the maximum threshold limit δ_j as this can only be handled by the repair mechanism to be discussed in section 5.5.

5.3 Neighbourhood Structure for the Local Searches (IDDIT)

Local searches' trajectories over a given search or solution space are unique; they traverse such a space by generating a neighbouring solution from an incumbent solution with the aim of either avoiding entrapment in a local optimum or in order to obtain a solution that is at least good and sometimes very close to an optimal one. However, there might be several ways in which such neighbouring solutions are generated. It should be noted that, a newly generated neighbouring solution does not necessarily satisfy all constraints; thus, one such idea is the continual stochastic generation of such neighbours until all incorporated constraints are satisfied. This idea is, without any doubt, inefficient and extremely costly due to the wastage of valuable time and other resources.

In order to avoid such wastages in our constrained PSP implementation, there is the need then to come up with an *intelligent* idea for generating neighbouring solution in order to achieve the aforementioned purposes. The idea we came up with was inspired by the three neighbourhood relations introduced in the work of Schaerf [129]. The three neighbourhood relations introduced therein are **idR** ([*i*]ncrease, [*d*]ecrease, [*R*]eplace), **idID** ([*i*]ncrease, [*d*]ecrease, [*I*]nsert, [*D*]elete) and **TID** ([*T*]ransfer, [*I*]nsert, [*D*]elete). Apart from some anomalies observed in the execution of these neighbourhood relations which we are going to exhibit later; we also notice some aspects of redundancy in their individual executions, and we argue that the operations involved in these neighbourhood relations can be effectively merged together in a single neighbourhood relation, while each operation can then be executed by introducing some probability weighting mechanism. For clarity of description and to easily showcase the identified loopholes, we hereby reproduce the first two neighbourhood relations as represented in Schaerf [129]:

idR ([i]ncrease,[d]ecrease,[R]eplace):

Description : The quantity of a chosen asset is increased or decreased. All other shares are changed accordingly so as to maintain the feasibility of the portfolio. If the share of the asset falls below minimum it is replaced by a new one.

Attributes : $\langle a_i, s, a_j \rangle$ with $a_i \in A, s \in \{\uparrow, \downarrow\}, a_j \in A$

Preconditions: $a_i \in L$ *and* $a_j \notin L$

Effects : If $s = \uparrow$ then $w_i = w_i \bullet (1+q)$, otherwise $w_i = w_i \bullet (1-q)$. All values w_k -ε_k are renormalized and not x_k to ensure that no asset rather than a_i can fall below the minimum.

Special cases : If $s = \downarrow$ and $w_i(1-q) < \varepsilon_i$, then a_i is deleted from L and a_j is inserted with $w_j = \varepsilon_j$. If $s = \uparrow$ and $w_i(1+q) > \delta_i$, then w_i is set to δ_i

Reference : Revised version of Chang et al. (2000).

Figure 16: idR neighbourhood definition

idID([i]ncrease, [d]ecrease, [I]nsert, [D]elete):

Description: Similar to idR, except that the deleted asset is not replaced and insertions of new assets are also considered.

Attributes: $\langle a_i, s \rangle$ with $a_i \in A, s \in \{\uparrow, \downarrow, \mapsto\}$

Preconditions: If $s = \downarrow$ or $\uparrow$ then $a_i \in L$. If $s = \mapsto$ then $a_i \notin L$

Effects: If $s = \uparrow$ then $w_i = w_i \bullet (1+q)$; If $s = \downarrow$ then $w_i = w_i \bullet (1-q)$; If $s = \mapsto$ then $a_i : \varepsilon_i$ is inserted into L. The portfolio is repaired as explained above for idR.

Special cases: If $s = \downarrow$ and $w_i(1-q) < \varepsilon_i$, then a_i is deleted from L and it is not replaced. If $s = \uparrow$ and $w_i(1+q) > \delta_i$, then w_i is set to δ_i

Figure 17: idID neighbourhood definition

For instance, consider the *special cases* under **idR** neighbourhood relation; We argue that when a decrease operation is executed, and consequently, the resultant weights are renormalized accordingly in order to add up to unity and maintain portfolio's feasibility; it is still possible to notice that, at least one of the constituent asset's weight *shoot* up beyond the maximum threshold, δ_i. In the same vein, we observed that, if an increase operation was executed, it is likely to have another asset's weight going below the minimum threshold after renormalization. Let us now present our argument by a simple numerical example as follows:

Suppose there are $n = 10$ assets in a given universe, U of assets $(i.e.\ U = \{A_1, A_2, A_3, A_4, A_5, A_6, A_7, A_8, A_9, A_{10}\})$ out of which no given portfolio is allowed to contain more than $k = 5$ assets. Let the minimum and maximum threshold limits for all assets weight be $\varepsilon_i = 1\%$ and $\delta_i = 70\%$ respectively. Furthermore, let us assume that, the step value, $q = 0.95$ (Refer to Figures 5.3(a) and 5.3(b)), and suppose a given portfolio, L composed of 5 assets: $(i.e.\ L = \{A_1, A_5, A_3, A_9, A_2\})$ whose corresponding actual proportion of portfolio funds are

$\{w_1 = 0.06,\ w_5 = 0.11,\ w_3 = 0.09,\ w_9 = 0.04,\ w_2 = 0.7\}$ respectively; notice that, all assets weights sum up to unity. Now according to the provisions in special cases of **idR** neighbourhood relation, if a ***decrease*** operator is executed on, say, asset A_1 $\left[i.e.\ w_1 = w_1\left(1-q\right)\right]$ and this implies $w_1 = 0.06\left(1-0.95\right) = \mathbf{0.003}$ which without any doubt is less than the minimum threshold of 1%; and consequently asset A_1 will be deleted. Now suppose the deleted asset A_1 is replaced by another asset A_7 (not already in L before) with weight equal to the minimum threshold of 1%; then the new composition of portfolio, L will now then be $\{A_7, A_5, A_3, A_9, A_2\}$ with corresponding weights $\{\boldsymbol{w_7 = 0.01},\ w_5 = 0.11,\ w_3 = 0.09,\ w_9 = 0.04,\ w_2 = 0.7\}$ adding up to 0.95. Now in order to maintain the feasibility of the resultant portfolio and at the same time ensure that weights add up to unity; the weights have to be renormalized thereby leading to a portfolio with new assets' weights values of: $\{w_7 = 0.010,\ w_5 = 0.116,\ w_3 = 0.095,\ w_9 = 0.042,\ \boldsymbol{w_2 = 0.737}\}$. A problem that immediately resurfaces in the resultant portfolio's weight configuration is what we are trying to make the reader be aware of; as it can easily be seen that, the weight of A_2 (in bold) now has a value greater than the maximum threshold of 70%.

In another example, suppose q = 0.05 and consider another portfolio L consisting of the same set of 5 assets $\{A_1, A_5, A_3, A_9, A_2\}$ as used in the previous example having corresponding weights: $\{w_1 = 0.02,\ w_5 = 0.01,\ w_3 = 0.24,\ w_9 = 0.04,\ w_2 = 0.69\}$; now let us suppose an ***increase*** operator is executed on asset A_2 $\left[i.e.\ w_2 = w_2\left(1+q\right)\right]$, its corresponding weight will now assume a new value of 0.7245 $\left(i.e.\ w_2 = 0.69\left(1+0.05\right) = \mathbf{0.7245}\right)$ which is apparently greater than the maximum

threshold value of 70%, and consequently (based on the provision of idR special cases) has to be capped to the same maximum threshold value. With this new value of w_2 being set to 70%, the weights do not add up to unity anymore and the portfolio consequently loses its feasibility, hence the need for renormalization. After the renormalization operation is executed, the new set of renormalized portfolio weights will now be: $\{w_1 = 0.0198,\ \mathbf{w_5 = 0.0099},\ w_3 = 0.2376,\ w_9 = 0.0396,\ w_2 = 0.6931\}$, thereby forcing w_5 to take a new value (in bold) under the minimum threshold of 1%.

These problems that resurfaced in our 2 simple examples after executing the decrease/increase operations followed by weights' renormalization proves the validity of our argument on the accompanying loopholes with the way those neighbourhood structures were implemented, and this further proves to us and the reader that, what the **idR** procedure currently provides in replacing an asset whose weight falls below the minimum threshold with another one as well as just capping the asset whose weight go beyond the maximum threshold is definitely not enough; the repair (renormalization only) mechanism is therefore flawed and thus, there is the need to do more by taking into account other marginal cases that might occur as a result of executing the aforementioned operations as outlined above. We believe the same argument can be drawn from the other two neighbourhood relations contained therein. In view of these, we proposed an efficient repair mechanism by adopting the repair strategy introduced in Chang *et al.* [20]. Our *iterative* repair mechanism (to be presented later in section 5.5) will always guarantee providing a feasible portfolio satisfying both minimum and maximum threshold constraints, while at the same time adding up to unity.

Apart from the problems that result from the loopholes observed above; we further noticed some redundancy in the operations of the three neighbourhood relations (**idR**, **idID** and **TID**), where we noticed for example, the main difference between especially **idR** and **idID** lies with the choice of either replacing or not replacing a given asset. Similarly, **TID** has some elements of insertion and deletion of an asset as in the case of **idID**, thereby rendering some operations redundant. In order to address these redundancy concerns, we propose a single neighbourhood structure that incorporates all the operations (Increase, Decrease, Delete, Insert and Transfer) obtainable in the three neighbourhood relations presented in Schaerf [129]; hence, the name IDDIT.

The neighbourhood structure introduced in this section is designed to work with our Local Search algorithms (SA, TS, Par_SA) for the implementation of the constrained PSP formulation. It involves introducing and executing five different neighbourhood operations with uniform probability of being chosen for execution. The five different operations include *increasing*, *decreasing*, *deletion*, *insertion* and *transfer*. Because in our formulation, the PSP variables are continuous; the concept of neighbourhood *move* in the first two operations (*increasing* and *decreasing*) involves the notion of moving within a given neighbourhood by using *step* (a real-valued parameter less than 1) multiplied by a uniformly distributed number randomly generated and lying in the interval (0, 1); the result of which is either added to/subtracted (as the case may be) from a given asset's actual portfolio proportion (weight), after which a renormalization of the entire portfolio weights is executed – a process meant to enforce the satisfiability of (an important) budget constraint. After several experimental trials we found that, in both cases a *step* value of 0.975 is found to work well with our problem.

As in Chang *et al.* [20], our neighbourhood structure allows each candidate solution (portfolio) to be characterized by two sets as described in section 5.1 above. The first set, denoted by L, is an integer set of asset indices; while the other denoted by W, is a set of real numbers signifying the actual proportion of portfolio funds for the corresponding assets in set L. We implemented our neighbourhood function in such a way that it processes begin by randomly generating an integer value (we termed as *decision index*) less than 5 $\left(i.e.\ decision\ index \in Z[0,4]\right)$ responsible for deciding which among the five neighbourhood operations to execute; in which a *decision index* of 0, 1, 2, 3 or 4 denotes the execution of ***increase***, ***decrease***, ***delete***, ***insert*** or ***transfer*** respectively. Next, we then randomly choose one or two asset's indexes (two assets in case of a transfer operation and one asset otherwise) from L upon which the selected operation is to be implemented.

In a given iteration, when the decision index takes a value 0, an ***increase*** operator is executed, the two sets (L and W) characterizing the solution of any given portfolio are passed to the repair structure (to be discussed later) responsible for maintaining portfolio's feasibility and constraints satisfaction. However, when any of the remaining four operations are executed, some checks and minor sub-operations are conducted before passing the portfolio to the repair procedure. For instance, when the *decision index* takes a value 1 and the ***decrease*** operator is executed on a randomly chosen asset in L; now, before passing the portfolio to the repair mechanism; the function first checks if the affected asset's actual weight falls below the minimum threshold, $\varepsilon_i, \left[i.e.\ If\ w_i\left(1 - q \times rand(0,1)\right) < \varepsilon_i\right]$, and if so, the function randomly generates a binary variable [0 or 1] which decides either to just delete or replace the affected asset. If the decision reached is to only *delete*; the function further checks if

there are at least two assets in the portfolio $\left(i.e.\ If\ |L| \geq 2\right)$ before the affected asset is deleted, otherwise, it is spared. On the other hand, if the decision reached is to *delete and replace*, the newly introduced asset assumes a weight whose magnitude is equivalent to the minimum threshold value, ε_i.

The ***delete*** operator which is normally executed when the *decision index* takes a value 2 randomly deletes an asset from a given portfolio L provided there are at least two assets in it, irrespective of whether its weight undershoots or overshoots the minimum or maximum threshold limits respectively; the corresponding asset's weight is then set to the minimum threshold value, ε_i. On the other hand, an ***insert*** operator is chosen for execution when the *decision index* takes a value 3. This operator, provided the cardinality of L is less than the maximum cardinality value of k $\left(i.e.\ |L| < k\right)$, allows for the insertion of an asset randomly chosen from set $\{U - L\}$ into the portfolio L with a portfolio proportion equal to the minimum threshold limit $\left(i.e.\ w_i = \varepsilon_i\right)$. Finally, a ***transfer*** operator whose *decision index* takes value 4 can be executed if and only if there are at least two assets in portfolio L, in which one serves as a 'donor' and the other a 'recipient'. A 'donor' asset deducts a portion of its weight with the aim of transferring the same to the lucky 'recipient'; if there are more than two assets in portfolio L, both (donor and recipient) assets are chosen randomly, and when this happens the portfolio's feasibility remains unaffected, especially after passing the results to the repair mechanism. The following pseudocode depicts how we implemented our neighbourhood structure:

```
Function IDDIT(k, ε, δ, L, w)
Begin {
    % N is the Universe of assets
    % L is the set containing at most K assets in the current solution
    % w_j is the actual proportion associated with asset j ∈ L
    randomly generate decisionIndex (integer ∈ [0,4])
    randomly generate an index j ∈ L
    If (decisionIndex = 0) then
        w_j = w_j × (1 + step × rand[0,1])
    Else if (decisionIndex = 1) then {
        w_j = w_j × (1 − step × rand[0,1])
        If (w_j < ε_j) then {
            If (|L| > 1) then {
                L = L − [j]
                w_j = 0.0    }
        }
    }
    Else if (decisionIndex = 2) then {
        If (|L| > 1) then    {
                L = L − [j]
                w_j = 0.0    }
    }
    Else if (decisionIndex = 3) then {
        generate a new index j ∈ N − L
        If (|L| < k) then  {
            L = L ∪ [j]
            w_j = ε_j        }
    }
    Else {
        If (|L| > 1) then    {
        generate a donor index i ∈ L
        generate a recipient index j ∈ L (j ≠ i)
        w_i = w_i − (w_i × 10%)
        w_j = w_j + (w_i × 10%) }
    }
    Call repairAssetWeights(K, ε, δ, L, w)
End
```

Figure 18: IDDIT (The neighbourhood structure for the local searches)

5.4 Neighbourhood structure for Swarm Algorithms

The neighbourhood structure described in the previous section is most suitable for our Local Search algorithms, and can in no way be applied (without making some modifications) to our swarm algorithms (PSO and SWAN) due to the rule behind their basic implementation procedure. For instance, a basic PSO algorithm involves dealing with several particles (solutions) that inter-communicate their search history and progress with one another, while at the same time keeping track of their personal best

ever found solution and that of the entire swarm. Furthermore, all particles update their current positions on the search/solution space according to some rule that imposes utilization of some information from the history of their trajectory and the global best particle.

One aspect that makes implementation of constrained PSP in especially PSO (and by extension, the SWAN) difficult is the concept of updating particles' positions. This is because, even if the search begins from a set of fully feasible solutions in which none is found to violate either or both sets of (cardinality and floor & ceiling) constraints; by the time particles undergo a velocity and particles' position updating mechanism, the new solution may well be infeasible; this scenario can easily be verified by analyzing how the pair of equations 3.5.5(b) and 3.5.5(c) operate; we hereby reproduce these equations below:

$$v_{ij}^{k+1} = wv_{ij}^{k} + C_1 R_{1,j}^{k}\left[lb_{ij}^{k} - x_{ij}^{k}\right] + C_2 R_{2,j}^{k}\left[gb_{j}^{k} - x_{ij}^{k}\right]$$

$$x_{ij}^{k+1} = x_{ij}^{k} + v_{ij}^{k+1}$$

The above challenge, we opined, is what led to the very minimal implementation of constrained PSP using PSO algorithm in comparison to other evolutionary algorithms (like GA) where the individuals (candidate solutions) *do what they want* on the search space without any need to track or utilize some information to do with their personal history or that of the best individual.

In the constrained implementation of PSO/SWAN, it is extremely difficult to update particles based on the conventional PSO update mechanism, due to the reasons stated above. In view of this challenge we propose an implementation devoid of any velocity

and position update mechanisms; however, we introduced a real-valued scoring term (which we referred to as *score velocity*) that will be used to allow particles to make an informed decision in moving to a neighbouring solution within their immediate neighbourhood, by tracking the success of their search history (such as personal best solution), as well as that of the entire swarm (global best). It should be understood that, although, the concept of moving towards either the personal or global best is extremely restricted (due to difference in cardinalities, and therefore operating in different search subspaces), the proposal we present allows each particle to make a decision of moving within its neighbourhood based on the information made available to it about the composition of those reference (personal and global best) solutions.

In this constraint formulation of PSO/SWAN; each particle should be viewed as being confined within its search subspace allowing it to search for better solutions within its immediate neighbourhood. While doing that, each particle also has the ability to inter-communicate with other particles and relate their search history as well as that of the entire swarm. We decide at this stage to introduce a neighbourhood structure (similar to IDDIT described in the previous section) that will allow particles to search their subspaces with a bit of guidance, unlike what is obtainable in the IDDIT structure, where all operations and sub-operations are entirely based on a random chance. The neighbourhood structure, occasionally allows a particle to 'jump' away from its subspace to another after some quite number of iterations; this phenomenon will eventually allow particles to converge at a global solution.

The neighbourhood relation developed, although similar to IDDIT has some remarkable enhancements. The similarity between the two lies in using the same

solution representation mechanism of defining a solution (portfolio) into two different sets of asset indices (denoted by L) and asset's weights (denoted by W also) as introduced in section 5.2. It also uses the same five neighbourhood operations (Increase, Decrease, Delete, Insert and Transfer) in executing a *move* from one current solution to its neighbouring solution. As part of the enhancements we introduced, this neighbourhood relation possesses a bit of *intelligence* by utilizing some fitness values that allow it to wisely decide which asset should be eliminated from a given portfolio, given that a ***delete*** operator is executed. It also smartly decides which asset to be chosen (from the remaining ones in the universe) and inserted into portfolio L, if an ***insert*** operator is executed. Let us now demonstrate how the neighbourhood structure works and how particles can move around their immediate neighbourhood and how the concept of particles' 'jump' can be justified. Let the reader refer to the following figure in the discussion that follows:

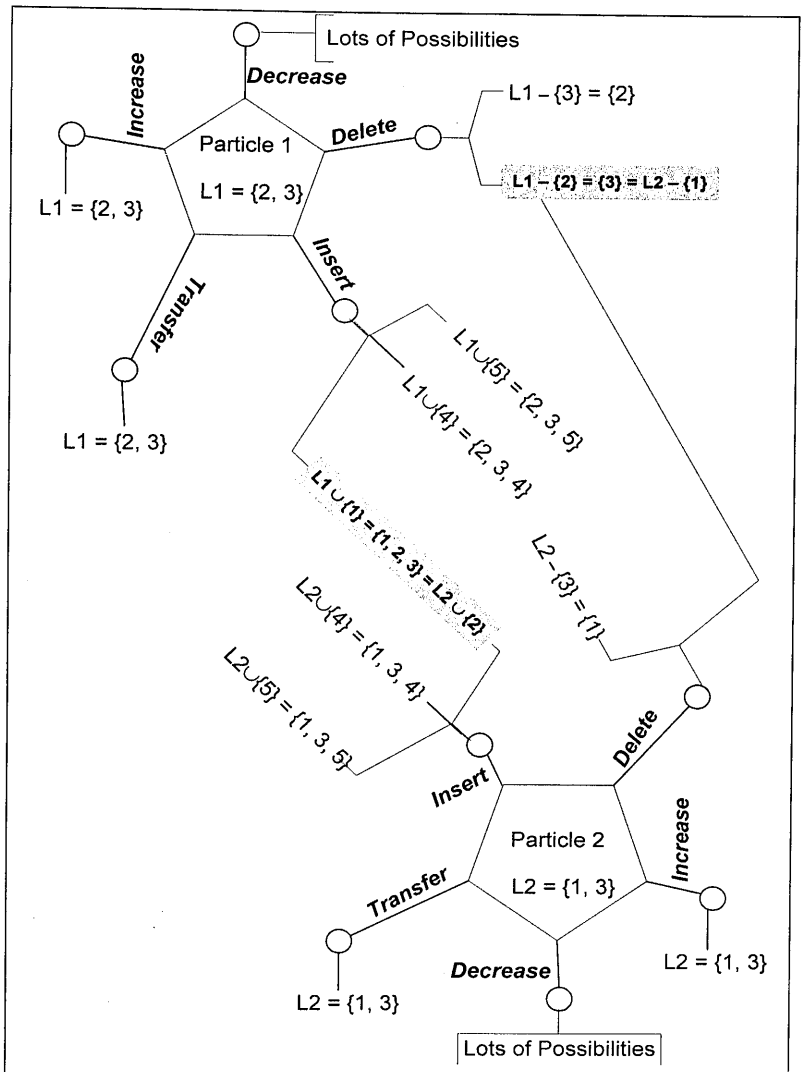


Figure 19: showing the concept of particles' move and jump

Suppose there are $n = 5$ assets in a given universe, U of assets $\left(i.e.\ U = \{1, 2, 3, 4, 5\}\right)$ out of which any given portfolio is allowed to at most $k = 3$ assets. Assuming particles 1 and 2 each has two assets in their current portfolio (where L1 = {2, 3} and L2 = {1, 3}); we now analyze how the two particles (*particle*1 and *particle*2) operate under the neighbourhood relation. It can be easily observed that, for both particles (and by extension all other particles) when the ***increase*** or ***transfer*** operator is executed, the portfolios' asset composition remains unchanged; and thus, the particle's search is then confined within its immediate neighbourhood. On the other hand, the execution

of any of the other three operators (***decrease***, ***delete*** and ***insert***) can quite often result in a portfolio with different asset composition; thereby enabling a particle to escape its immediate neighbourhood and 'jump' over to another. The continuous jumping from one subspace to another (by particles) will eventually lead them towards convergence to the best solution.

To illustrate what we mean by 'jumping' to another search subspace region; suppose a ***delete*** operator is executed on particle 1 ($L1 = \{2, 3\}$) and that asset 2 was marked for elimination (i.e. $L1 - \{2\}$), the resultant portfolio will now consist of only asset 3 (i.e. $L1 - \{2\} = \{3\}$). This particle will find itself searching for better solutions in the same neighbourhood as particle 2 ($L2 = \{1, 3\}$) when a ***delete*** operator acted upon it, and asset 1 is eliminated (i.e. $L2 - \{1\} = \{3\}$), as can be seen in the upper gray-colored rectangle on Figure 16. Similarly, if an ***insert*** operator introduced asset 1 in particle 1's portfolio composition (i.e. $L1 \cup \{1\}$); the resultant portfolio will now consist of three assets (i.e. $L1 \cup \{1\} = \{1, 2, 3\}$); which is equivalent to the same search space region in which particle 2 will find itself on, if another ***insert*** operator introduces asset 2 into its current configuration (i.e. $L2 \cup \{2\} = \{1, 2, 3\}$) as can be seen in centermost gray-colored rectangle.

The neighbourhood structure uses an evolvable real-valued number we referred to as *score velocity* which a given portfolio uses to communicate to the global and personal best solutions. The same *score velocity* is used to compute a *fitness* value that will be used in either deleting/inserting an asset from/into a given portfolio *L*. The ***delete*** operator is executed, so that the asset with the lowest *score velocity* stands a higher chance of being eliminated from the portfolio; while as for the ***insert*** operator, assets

with higher *score velocity* (from the remaining ones in the universe) stands a higher chance of being included into the portfolio, L. In the beginning of the search, the *score velocity* for each particle is assigned a uniform value of 1.0 to all assets in the universe of assets $\left(i.e.\ scoreVel_i = 1.0 \mid \forall i \in U\right)$; but if an asset is one of the constituents of the personal best solution, its *score velocity* increases by some fraction of the total cumulative *score velocity* of the constituent assets in L; similarly, if it is part of the assets that makes up the global best solution, its *score velocity* further increases by some fraction of the current total cumulative *score velocity* of the current portfolio L. The following two conditions illustrate how a *score velocity* of a given asset in a portfolio, L evolves as the search continues:

$\forall p \in swarm\ of\ particles$
  $\forall i \in L^p \{$
    If $\left(i \subset local\ best\ solution\right)$ **then**
      $SV_i^p = SV_i^p + 3\% \times \left(total\ cummulative\ score\ velocity\ for\ L^p\ members\right)$
    If $\left(i \subset global\ best\ solution\right)$ **then**
      $SV_i^p = SV_i^p + 5\% \times \left(total\ cummulative\ score\ velocity\ for\ L^p\ members\right)$
  $\}$
$\}$

Figure 20: Updating particles' score velocity

This simply means, an asset which forms part of both the personal and global best solutions receives 'double reward'; thereby maximizing its chances of being included in a portfolio when an ***insert*** operator is executed, provided it is not already part of the current portfolio L; similarly, such an asset has a minimum chance of being eliminated from a current portfolio, L when the ***delete*** operator is executed.

The reader might be tempted to ask, then: "How can that be done?" To answer this question, it is important if we make the reader understand that, any given asset in the universe $\left(i.e.\ asset\ i \mid \forall i \in U\right)$ can only be a member of one of the two subsets in the

universe $(i.e.\{L\} or \{U-L\})$ at any given point in time. For all assets in portfolio L, we decided to compute the *inverse* of their various *score velocities* after which we then compute the *cumulative* ***inverse*** *score velocity*; while for all assets in the other subset $\{U - L\}$; we decided to compute their *cumulative score velocity* which helps in choosing which asset to include in a given portfolio.

The following pseudocode shows how we are able to part the universe of assets into two distinct sets $\{L\}$ and $\{U-L\}$, after which we compute the parameters that will help us accomplish a move to a neighbouring solution.

Function ***computeSVParameters***$(scoreVel,\ L,\ Q, CummSVQ_i, InvCummSVL_j)$
Begin {
% U is the Universe of assets
% L^p : set containing at most k assets in the *particle* $p's$ current solution
% Q^p: set of all assets in the univese, U, not in portfolio L^p $(\text{i.e. } Q^p := \{U - L^p\})$

$Q^p := \varnothing$ % initialize Q to a null set
For all $p \in swarm$
For all $i \in U$, If $i \notin L^p$ then $Q^p := Q^p \cup [i]$ % partition the Universe into L^p & Q^p
$totCummSVQ^p := 0$; % initialize the fitness for Q^p members
For all $i \in Q^p$ **do** {
$totCummSVQ^p := + scoreVel_i$
$CummSVQ_i^p := totCummSVQ^p$ }
$totInvCummSVL^p := 0$; % initialize the fitness for L^p members
For all $j \in L^p$ **do** {
$totInvCummSVL^p := + 1/(scoreVel_j)$
$InvCummSVL_j^p := totInvCummSVL^p$ }
}**End**

Figure 21: Computing score velocity parameters

The function in Figure 21 above accepts portfolio L and the *score velocity* vector as input; while at the same time generating Q, *CummSV* and *InvCummSV* as output variables. These variables evolve as the program progresses and different portfolios are generated. *CummSV* is especially used when an ***insert*** operator is executed; it gives

a higher chance of selection to an asset with high *score velocity* which is to be inserted into a candidate portfolio. *InvCummSV* on the other hand, favours the asset with lower *score velocity* for elimination when a ***delete*** operator is to be executed.

We now present the pseudocode of the program function used in implementing a move around the immediate neighbourhood of a given particle.

Function *swarm* ***IDDIT*** $(k,\ \varepsilon,\ \delta,\ scoreVel,\ L^p, w^p)$
Begin {
% *U is the Universe of assets*
% L^p : *set of at most k assets indices for particle p's portfolio L*
% Q^p : *set equivalent to* $\{U - L^p\}$
% w_j^p *is the actual proportion associated with asset* $j \in L^p$
% $totInvCummSVL^p$ *is the total inverse cummulative score velocity for* L^p
% $totCummSVQ^p$ *is the total cummulative score velocity for* Q^p
call ***computeSVParameters*** $(scoreVel,\ L^p,\ Q^p,\ CummSVQ_i^p,\ InvCummSVL_j^p)$
For all $p \in$ *numparticles* {
randomly generate decisionIndex (*integer* $\in [0,4]$)
randomly generate an integer index $j \in L^p$
If $(decisionIndex = 0)$ **then** $\{w_j^p = w_j^p \times (1 + step \times rand[0,1])\}$
Else if $(decisionIndex = 1)$ **then** {
$w_j^p = w_j^p \times (1 - step \times rand[0,1])$
If $(w_j^p < \varepsilon)$ **then** { **If** $(|L^p| > 1)$ **then** $\{L^p = L^p - [j]\ \&\ w_j^p = 0.0\}\}\}$
Else if $(decisionIndex = 2)$ **then** {
$val = rand[0,1] \times totInvCummSVL^p$
If $(val \le InvCummSVL_0^p)$ **then** { **If** $(|L^p| > 1)$ **then** $\{L^p = L^p - [L_0^p]\}\}$
Else {
$(j > 0 \mid \forall j \in L^p)$
If $(val > InvCummSVL_{(j-1)}^p)\ \&\ (val \le InvCummSVL_j^p)$ **then** {
If $(|L^p| > 1)$ **then** $\{L^p = L^p - [L_j^p]\}\}\}\}$
Else if $(decisionIndex = 3)$ **then** {
$val = rand[0,1] \times totCummSVQ^p$
If $(|L^p| < k)$ {
If $(val \le CummSVQ_0^p)$ **then** { $\{L^p = L^p \cup [Q_0^p]\}\ \&\ \{w_0^p = \varepsilon\}\}$
Else { $(i > 0 \mid \forall i \in Q^p)$
If $(val > CummSVQ_{(i-1)}^p)\ \&\ (val \le CummSVQ_i^p)$ **then** {
$\{L^p = L^p \cup [Q_i^p]\}\ \&\ \{w_i^p = \varepsilon\}\}\ \}\ \}\ \}$
Else {
If $(|L^p| > 1)$ **then** {
randomly generate a donor $(i \in L^p)\ \&$ *recipient* $(j \in L^p \mid j \ne i)$ *indices*
$w_i^p = w_i^p - (w_i^p \times 10\%)\ \&\ w_j^p = w_j^p + (w_i^p \times 10\%)\ \}\}$
Call repairAssetWeights$(k,\ \varepsilon,\ \delta,\ L^p, w^p)$
}*End*

Figure 22: Swarm techniques neighbourhood definition

From Figure 22 above, before the program moves into the For-loop of the different particles, it has to first invoke/call the score velocity parameter function, in order to make available some parameters (*InvCummSVL*, *CummSVQ*, *totInvCummSVL*, *totCummSVQ*) that will be found important for the successful implementation of the operations. The superscript in most of the identifiers signifies a particle; while the subscript identifies a given dimension. It can be seen also that, our swarm neighbourhood structure is in many aspects similar to the IDDIT neighbourhood structure for the local searches defined in Figure 18. The major difference between the two, has to do with how the ***delete*** and ***insert*** operators are executed. For instance, when the decision index takes the value 2 (delete operation), the function – swarmIDDIT, randomly generates a real value, *val*, smaller than or equal to the *total inverse cumulative score velocity* (*totInvCummSVL*) of all assets in a given portfolio *L*; if this value falls between 2 consecutive *InvCummSVL* values, the asset with higher *InvCummSVL* value is eliminated from the portfolio, provided there is more than one asset in that portfolio, otherwise the asset is spared.

On the other hand, when the insert operator is to be executed, there is the need to randomly generate a real value, *val*, whose magnitude is at most equal to the *total cumulative score velocity* (*totCummSVQ*) of all assets in the universe who are not members of *L* (*Q* = *{U-L}*). As in the case of a delete operation, this value is positioned between two consecutive *CummSVQ* values, in which an asset with the highest *cumulative score velocity* (*CummSVQ*) is eliminated from *Q* and inserted into portfolio *L* with corresponding weight equivalent to the minimum threshold; provided there is room for insertion of an additional asset (i.e $|L| < k$).

5.5 The Repair Mechanism

It is obvious that, executing any of the five operations described in our neighbourhood structures defined in sections 5.3 and 5.4 above will definitely distort the feasibility of a given solution. For example, this distortion occurs when the proportion of one of the constituent assets in any portfolio (as described in section 5.1) is either *increased* or *decreased* by a small (*step*) value in the interval (0, 1). Similarly, when an asset and its corresponding portfolio proportion are *deleted* or a new one is *inserted* into the portfolio, there is the need to *repair* the assets' weights in order to satisfy the budget as well as the floor & ceiling constraints. It should be understood that, the constraint relating to the minimum proportion limit ε_j can be easily satisfied (by all assets in portfolio L) in a single iteration as described in section 5.2 above. However, an iterative mechanism would definitely be required to ensure that constraints relating to the upper proportion limit δ_j are satisfied also. Our repair approach is adapted from the approach used in Chang *et al* [20] which ensures an effective handling of constraint violation. In view of this, we hereby present our repair mechanism in the following pseudocode:

```
FUNCTION repairAssetWeights(K, ε, δ, L, z, w)
Begin{
    % U is the universal set containing all assets
    % L is the set containing at most k assets in a given portfolio
    % w_j is the actual proportion associated with asset j ∈ L
    % z_j is a binary variable taking value 1 for all assets j ∈ L and 0 otherwise
    A = Σ_{j∈L} ε_j                          % A is the accounted portfolio proportion
    F = (1 − Σ_{j∈L} ε_j)                    % F is the free portfolio proportion
    Sum_w = Σ_{j∈L} w_j                      % Sum_w is the sum of the unnormalized weights
    w_j = ε_j + (w_j / Sum_w) × F

    M = ∅                                    % M is a set of assets j whose weights are pegged at δ_j
    set Infeasible = True
    While Infeasible Do {
        set Infeasible = False
        While ∃ j ∈ L − M with w_j > δ_j Do {  % If any w_{j(∀j∈L−M)} exceeds δ_j
            L = L − [j]                      % Remove asset j from L
            M = M ∪ [j]                      % Insert asset j into M
        }
        Sum_w = Σ_{j∈L−M} w_j                % The updated sum of the weights
        F = 1 − (Σ_{j∈L−M} ε_j + Σ_{j∈M} δ_j)   % The updated free portfolio proportion
        w_j = δ_j  ∀j ∈ M                    % All weights of assets in M are set to δ_j
        w_j = ε_j + (w_j / Sum_w) × F  ∀j ∈ L − M   % Updated weights might violate constraints
        If ∃ j ∈ L − M with w_j > δ_j Then    % If some weights violate some constraints
            Infeasible = True                % Goto outer While – Do loop
    }
    ∀i ∈ L, z_i = 1  & ∀i ∈ {U − L}, z_i = 0;
    ∀i ∈ U, w_i = w_i × z_i
}
End
```

Figure 23: An effective repair approach for constrained PSP formulation

By taking a critical look at our iterative repair method in Figure 23 above, the reader can easily notice that, the repair mechanism ensures that all assets weights satisfy both set of constraints (the budget and the floor & ceiling constraints).

5.6 Results and Evaluation

For the constrained case, we decided to test and evaluate our algorithms by solving different constrained cases, in which portfolios are restricted to have no more than a specific number (also known as cardinality) of assets composition as well as making sure that any constituent asset has at least the minimum allowable proportion of the

portfolio funds invested in it. Recall that, in our unconstrained implementation (Chapter 4); our high expectations of GA's performance were unfounded, when surprisingly its performance was found to be poor in comparison to other EAs (such as PSO and SWAN); and because of this, we decided not to implement a constrained version of GA unless we are able to sort out what went wrong in its implementation. In view of this, we run the constrained implementation of the remaining 5 algorithms under different combinations of cardinality and floor & ceiling constraints. We first run our algorithms in which any candidate portfolio is restricted to have a maximum cardinality of only 10 assets under 3 different set of minimum threshold limits of 1%, 10% and 20%. Although, there were no explicit ceiling constraints, some were implied by the other (cardinality and floor) constraints. In the second set of experimental runs we scaled down the maximum cardinality value to 5 with similar combinations of minimum proportion limits (0.01, 0.1 and 0.2) as above (i.e. $K \leq 10$, 5 and $\varepsilon_j = 1\%$, 10% and 20%). We conducted a total of 60 simulation runs (5 algorithms × 2 datasets × 2 cardinality values × 3 minimum threshold limits) generating a total of (60 × 200) 12000 different points on the various heuristics constrained efficient frontiers.

5.6.1 Results:

This part is aimed at showing the various graphical representations of the results generated by our algorithms. It is important to make it known at this point that, although we intend to present the constrained results generated by the solver; we were unable to do so, due to the 'fear' of unfair comparison against the solver. This is because, the solver (being an exact solution provider) was not able to find solutions at some higher values of target return for both datasets, as it returns an empty output; hence, unable to determine any solution (be it optimal or otherwise) unlike our

algorithms which do produce solutions; even if these solutions are non-optimal, they are far better than no solution at all. Because of this, it will be hard to objectively compare our algorithms and the solver in the constrained cases. We thus, then decided to compare the algorithms among themselves.

The following figures shows the various EFs generated by our algorithms in solving the constrained case using two different data sets of varying sizes.

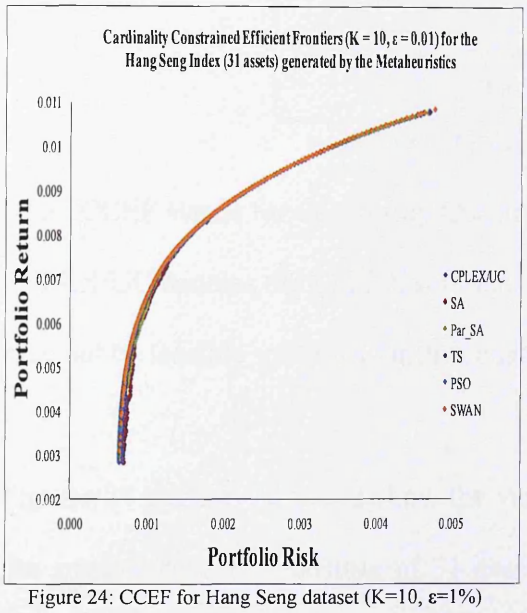


Figure 24: CCEF for Hang Seng dataset (K=10, ε=1%)

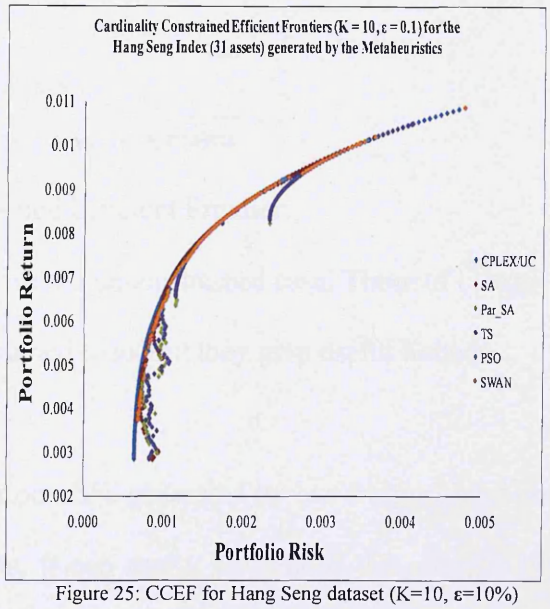


Figure 25: CCEF for Hang Seng dataset (K=10, ε=10%)

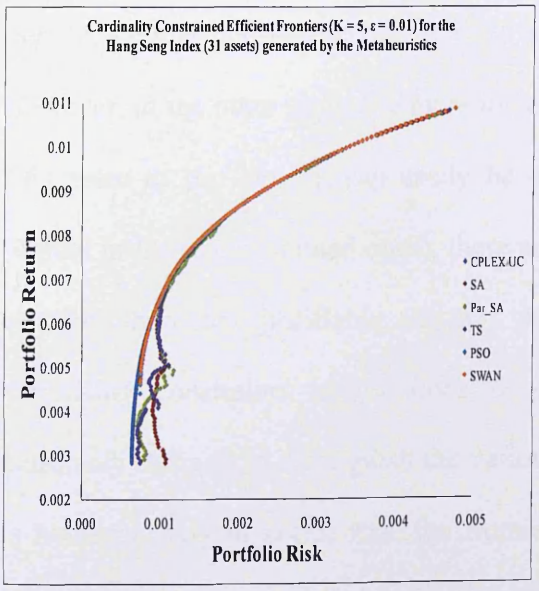


Figure 26: CCEF for Hang Seng dataset (K=5, ε=1%)

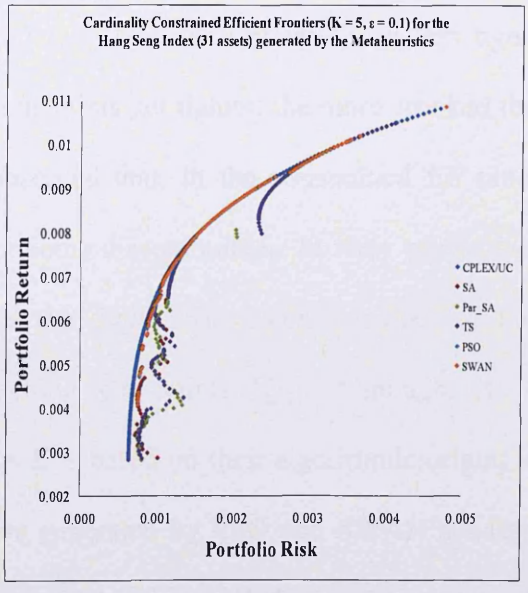


Figure 27: CCEF for Hang Seng dataset (K=5, ε=10%)

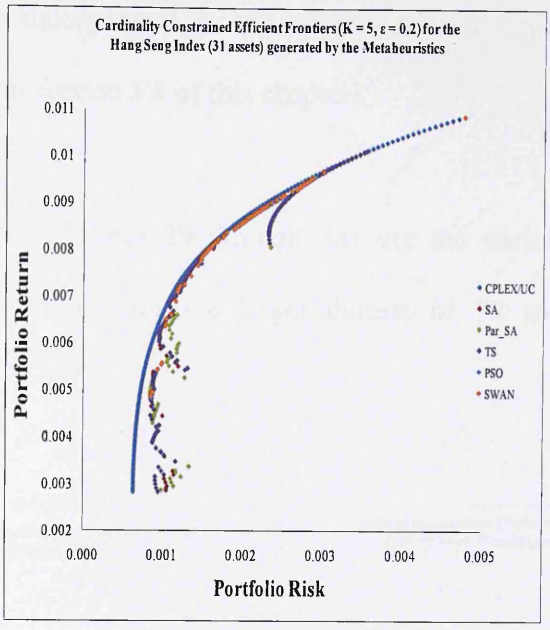


Figure 28: CCEF for Hang Seng dataset (K=5, ε=20%)

Note: CCEF stands for Cardinality Constrained Efficient Frontier.

CPLEX/UC denotes the CPLEX solutions for the unconstrained case. These of course may not be feasible solutions for the constrained case, but they give useful bounds.

Figures 24 through 28 above show the various *EF*s generated by our 5 algorithms for the smaller dataset consisting of 31 assets. It can easily be noticed that, Figure 24 looks smoother than others in this category, because its combination of cardinality and floor & ceiling constraints (i.e. $K \leq 10$ and $\varepsilon_j = 1\%$) are comparatively less tight. However, in the other plots, the more the constraints get tighter, the more crooked the *EF*s seem to be. Also, it can easily be observed that, in the constrained *EF* plots (unlike in the unconstrained ones), there are some discontinuities. In view of this, we can say these are justifiable, as they are the direct effects/consequences of the cardinality constraints and/or floor & ceiling constraints [20]. Although, it is extremely difficult to distinguish the various *EF*s based on their algorithmic origin; it is however, easy to notice that, the frontiers generated by PSO and SWAN are less crooked and have discontinuities; hence, more stable. This, we believe, is the 'fruit' of

a bit of *intelligence* incorporated in the neighbourhood structure for the swarm algorithms discussed in section 5.4 of this chapter.

The next sets of plots {Figures 29 through 33} are the various efficient frontiers generated by our algorithms for the larger dataset of 78 assets originally from FTSE100 index.

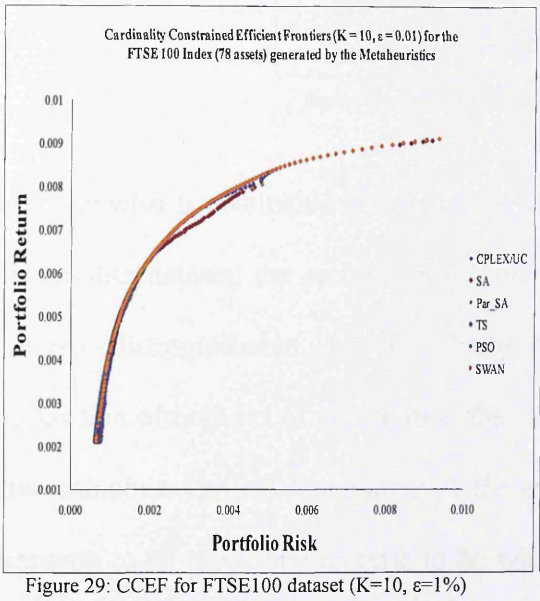


Figure 29: CCEF for FTSE100 dataset (K=10, ε=1%)

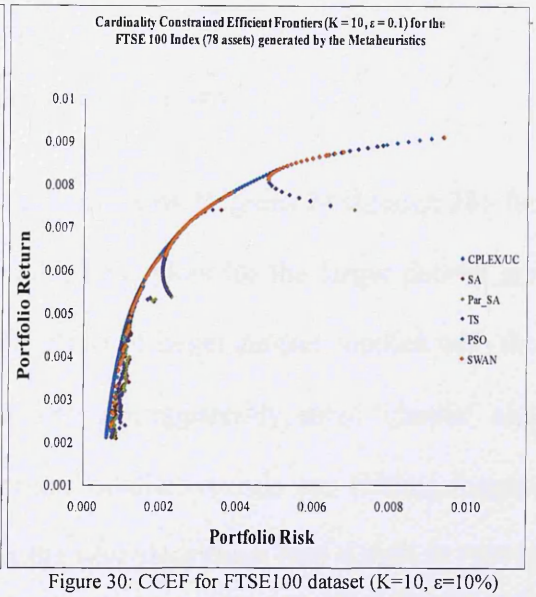


Figure 30: CCEF for FTSE100 dataset (K=10, ε=10%)

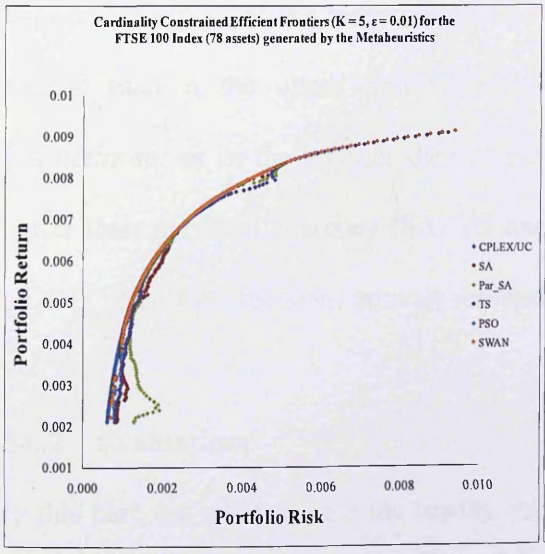


Figure 31: CCEF for FTSE100 dataset (K=5, ε=1%)

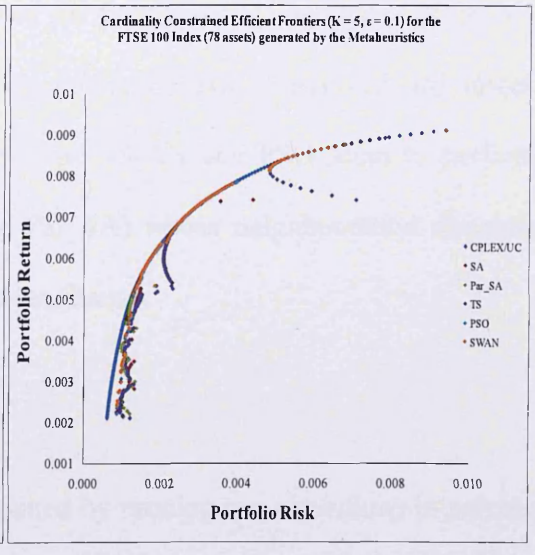


Figure 32: CCEF for FTSE100 dataset (K=5, ε=10%)

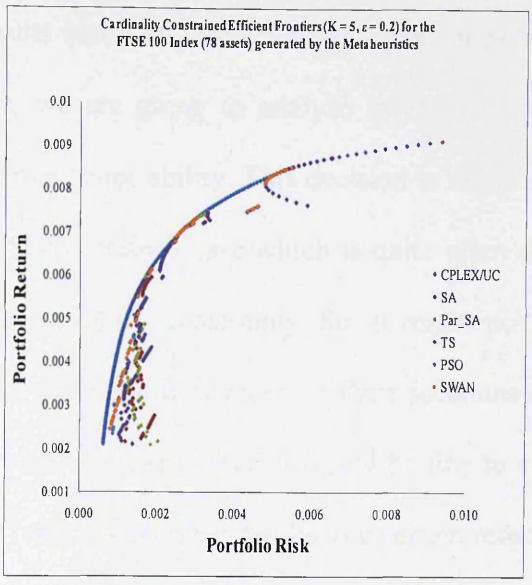


Figure 33: CCEF for Hang Seng dataset (K=5, ε=20%)

Just like what is obtainable in the first set of five *EF* plots {Figures 24 through 28} for the smaller dataset; the second set of constrained *EF* plots for the larger dataset are almost indistinguishable. However, due to the effect of larger dataset coupled with the imposition of both set of constraints; the *EF* plots are, apparently, more 'chaotic' and discontinuous. Critical observation of the various frontiers reveals one striking feature common to all plots; and this has to do with the way algorithms find it hard to 'get it right' at the bottommost part of the frontiers. It should be noted that, this should be expected, because at the bottommost side of the frontiers more and more assets are needed than at the upper part where the number of assets involved are fewer. Furthermore, as in the smaller dataset plots, the SWAN and PSO seem to perform better than the local searches (SA, TS and Par_SA) whose neighbourhood structure (IDDIT) based its decisions entirely on random chance.

5.6.2 Evaluation:

In this part, we will evaluate the results obtained by running our algorithms in solving the constrained implementation of the PSP. Unlike in the unconstrained formulation,

where we analyze results based on three evaluation measures (convergence, coverage and uniformity), here, we are going to analyze the performance of the algorithms based on only their convergence ability. This decision is informed by the nature of the results generated in the constrained case which is quite often discontinuous and non-uniform due to the effect of the constraints. So, it might not make much sense, to evaluate the algorithms based on how many of their solutions are found to dominate other algorithms and vice versa. However, it would be fine to evaluate them based on the proximity of their (entire) generated results to a certain reference point/solution. In view of this, we evaluate how well an algorithm does in terms of returning a very good solution by measuring (as in the unconstrained case) the area between the approximate constrained frontier generated by our methods and the exact (optimal) efficient frontier returned by the CPLEX solver for the unconstrained case. This enables us to access how good a particular heuristic constrained frontier is; as the closer it is to the optimal one the better. In order to accomplish this, we decided to measure the *mEds* of the various solutions generated by different algorithms in relation to the exact unconstrained solution generated by the nonlinear quadratic optimization solver (CPLEX 11.2) as done in chapter 4 dealing with the unconstrained case.

The following table gives a summary of the *mEds* and the average time taken (in seconds) by the algorithms to arrive at a solution in this constrained case.

Index (Assets)	k	ε	Methods	*mEd* ($\times 10^{-4}$)	Average Time (secs)
Hang Seng (31)	**10**	0.01	SA	0.3962	4.98
			ParSA	0.2709	7.76
			TS	0.3401	4.82
			PSO	0.1812	5.47
			SWAN	0.1649	5.91
		0.10	SA	1.5103	3.89
			ParSA	1.2016	4.07
			TS	1.4001	2.71
			PSO	1.0904	3.07
			SWAN	0.9243	3.21
	5	0.01	SA	2.1108	2.11
			ParSA	1.8113	3.59
			TS	2.0013	2.12
			PSO	1.2411	2.01
			SWAN	1.1812	2.09
		0.10	SA	2.9223	2.67
			ParSA	2.1581	4.44
			TS	2.4276	3.28
			PSO	2.0003	3.87
			SWAN	1.9053	3.91
		0.20	SA	6.4209	4.59
			ParSA	5.7657	5.64
			TS	6.1312	4.03
			PSO	4.8413	6.01
			SWAN	4.5641	6.24

Table 5: Showing the *mEds* and the average time (in secs) for the Hang Seng dataset

From Table 5 above, it can easily be observed that, the *mEds* values are (in all cases across the various algorithms) close to zero indicating that the distance between the reference (unconstrained) *EF* and those generated by our algorithms is very small. It should be recalled (from section 4.6.1) that, smaller values of *mEd* are much desired. It can also be noticed that, the more the constraints get tighter, the more the performance deteriorates. For instance, when the cardinality is set to 10 and the minimum allowable proportion is set to 1%, almost all the algorithms performed quite well, producing *mEds* values to the tune of 10^{-5}. One other thing worthy of noting and mentioning is the time (in secs) that algorithms take on average to return a solution. It can be

observed that, in comparison to the results obtained in chapter 4, the algorithms take much time; this we believe, is not unconnected with the effect of the imposed constraints.

Index (Assets)	K	ε	Methods	*mEd* ($\times 10^{-4}$)	Average Time (secs)
FTSE 100 (78)	10	0.01	SA	1. 4421	7.13
			ParSA	0.7745	10.87
			TS	0.9487	6.91
			PSO	0.6945	8.18
			SWAN	0.6420	8.54
		0.10	SA	2.9671	6.99
			ParSA	2.0173	9.74
			TS	2.5114	6.21
			PSO	1.3915	7.49
			SWAN	1.3055	7.97
	5	0.01	SA	2.9915	5.71
			ParSA	1.9349	6.04
			TS	2.6228	5.73
			PSO	1.4211	6.94
			SWAN	1.3852	7.18
		0.10	SA	4.7653	4.27
			ParSA	3.8365	9.10
			TS	4.4113	4.01
			PSO	1.9957	7.91
			SWAN	1.7616	8.28
		0.20	SA	5. 8353	7.19
			ParSA	4.5703	12.45
			TS	5.0009	6.01
			PSO	3.8914	8.81
			SWAN	3.7340	9.15

Table 6: Showing the mEds and the average time (in secs) for the FTSE100 dataset

Table 6 above further reveals the difficulty inherent in solving the constrained cases, especially when dealing with larger datasets. There is an apparent proof of performance deterioration and the average time taken seems to be growing at an exponential rate. The Tables 5 and 6 above further supports the superiority of the (guided) neighbourhood structure (for the swarm algorithms) described in section 5.5

over the (unguided) IDDIT neighbourhood structure described in section 5.4. This is so, because our swarm algorithms (PSO and SWAN) seem to produce a more stable EF plots than their local search counterparts (SA, Par_SA and TS) whose *EF*s seem more discontinuous and chaotic. In the same vein, despite the growth in dataset size coupled with constraints imposition, the *mEd* values for both PSO and SWAN remain consistently lower when compared to those produced by the trio of SA, Par_SA and TS.

5.7 Comparison with previous results

The Portfolio optimization problem as one of the most widely studied research areas in finance has been studied in several researches in which a wide range of models have been introduced or improved upon. In many of those researches different formulations involving lots of different objective functions, varying constraints sets, and different variable definitions have been proposed. In many of these studies, for instance; it is not uncommon to find several authors in different articles giving much credit to their findings as well as claiming superiority and advantages of their models and methods over others. But in many of such situations, it is almost impossible to fully justify such assertions, due to several factors that need to be taken into consideration: algorithmic implementations differ, datasets may be different, loopholes in actual implementation rarely reported, performance measures might be different, implementation platforms (such as systems' capacities) varies, and furthermore, a given model might do well over a given dataset it was tested upon without any assurance of robustness to other different datasets. Other researches, we must admit, are aimed at pursuing different goals, such as testing the efficiency and/or effectiveness of algorithms [25, 26, 129, 152, 155]; others aim to discard the conventional problem formulation in order to

develop a model alleged to be more suitable and appropriate than the standard one [91, 10, 49]; and others might be interested in developing a commercially viable model with good performance in order to help in professional decision making activities. All these factors can lead to wrong or biased comparisons and conclusions, hence, fair comparison among different works by different authors might be difficult.

5.7.1 Comparison with Chang *et al*

One of the only two datasets used in this research was originally used by Chang *et al* [20] and made publicly available by one of the authors at the OR Library [117]; so, it would really be desirable if we are able to compare our results with Chang *et al*'s. However, due to some of the issues raised above, we may not be able to do so. For instance, when one considers the work of Chang *et al*, it can be realized that, after including the expected return constraint into the objective function; they also introduced a weighting parameter $\lambda(0 \le \lambda \le 1)$ which when continuously varied (increased/decreased) will be used to trace the efficient frontier. Chang *et al*'s model is of the form:

Minimize $\lambda\left[\sum_{i=1}^{n}\sum_{j=1}^{n} w_i \sigma_{ij} w_j\right] - (1-\lambda)\left[\sum_{i=1}^{n} w_i \bar{r}_i\right]$

Subject to

$$\sum_{i=1}^{n} w_i = 1$$

$$\sum_{i=1}^{n} z_i = K$$

$$\varepsilon_i z_i \le w_i \le \delta_i z_i, \qquad i = 1,...,n$$

$$z_i \in [0,1], \qquad i = 1,...,n$$

Although, this (weighting) approach will allow for gaining more information (as they claimed) on some portions of the constrained frontier; its major implication involves

its inability to trace out the entire *EF*, and consequently rendering some portions invisible.

Another thing to consider is that the weighting approach does not produce an equally-spaced (i.e. homogeneously distributed) points/portfolios on the *EF*, thereby making it difficult for point-to-point comparison with our results, in which the problem is solved by considering different instances (of equally-spaced) values of target return. In addition, the results for the constraint case solved with the weighting approach were not provided; hence, the results cannot be reproduced and/or compared. Furthermore, their cardinality constraint was formulated with equality rather than with an inequality as in our case.

5.7.2 Comparison with Schaerf

If there is any work that we would be delighted to compare our results with, it should be no other than Schaerf's [129]. Recall that, our newly developed neighbourhood structure (IDDIT) and even some part of our swarm neighbourhood definitions were inspired by the neighbourhood relations contained therein. However, due to some reasons, some of which were raised above, we could not achieve that either. The major obstacle hindering us from comparing our results with Schaerf's is that, their results were not made publicly available as in the case of Chang *et al*'s unconstrained case. Instead of providing the results of their cardinality constrained solutions provided by their neighbourhood structures; their main concern was to compare the performance of the different neighbourhood definitions among themselves under different parameter settings. At a point, they had to admit: "*Given that the constraint problem has never*

been solved exactly, ***we cannot provide*** *[emphasis mine] an absolute evaluation of our results...*"

In their conclusion remark, they further state "*We solved public benchmark problems, but unfortunately no comparison with other results is possible at this stage.*"

6.0 Testing Algorithms on other problems

6.1 Some Applications

In this chapter we aim to explore the potential of our designed algorithms in some optimization applications other than the PSP. We intend to subject our algorithms to some tests of robustness in other academic researches such as function optimization in order to observe how well they adapt to other problems. However, it should be clearly understood that, this chapter is not primarily aimed at comparing our results to those of other previous researches with the hope of outperforming them; but rather to have a glimpse of how far our algorithms can go in optimizing standard test functions (both constrained and unconstrained) with known global optimum (i.e. how close to the global solutions our algorithms will be able to reach). In view of this, we selected the two best performing algorithms (PSO and SWAN) for testing their capability in solving the standard test functions as reported in the literature. In other parts of this chapter also, we intend to analyze the results obtained by our algorithms using some standard statistical techniques such as test of statistical hypothesis.

6.2 Global Optimization

Searching for an optimal state or configuration is one of the fundamental principles in many aspects of our life. This search begins in the microcosm when very tiny particles (such as atoms) join with one another (to form molecules) in order to minimize the energy in their electrons. Molecules also bond with one another to form solid bodies through the processes of freezing, and by so doing; they assume an optimal crystalline structure having the minimum energy.

In the same vein, several aspects of our life revolve around finding an optimum setting with the least effort possible. For instance, when we plan to travel for holidays we want to have an utmost enjoyment at the lowest possible cost. The same goes for flight tickets, in which we prefer to have 'first class' treatment at lower costs. Many business organizations thrive by making or rather maximizing their profit, while at the same time being alert to cut unnecessary costs; even investors (conventionally) would prefer to invest in a collection of assets that will result in maximizing their profit/return at the lowest possible risk. When we consider the field of Engineering; designers always prefer to maximize the performance of their designed products, while at the same time trying every way possible to minimize costs. With the above few examples, it is obvious that, studies in optimization are of immense importance, in which both our scientific interests and practical implications stand to benefit more from such studies. Before we continue with our analysis on how well our designed algorithms perform in respect of optimization on some standard test functions; there is the need to provide some definitions of some basic concepts.

6.2.1 Mathematical Optimization

It should be understood that, it is quite often possible to formulate any real world problem with a certain objective into a corresponding optimization problem; and all optimization problems can be represented in an explicit generic form provided they have an explicit objective. Therefore, according to Boyd and Vandenberghe [13], a mathematical optimization problem or simply, optimization problem takes the form:

$$\begin{aligned} &\underset{x\in\Re^n}{\text{Minimize}}\ f(\boldsymbol{x}) \\ &\text{Subject to} \\ &\qquad \phi_r(\boldsymbol{x}) \le b_r, \quad (r = 1,2,...,m), \end{aligned}$$

Where the components x_i of the vector: $\boldsymbol{x} = (x_1, x_2, ..., x_n)^t \in \Re^n$ are referred to as optimization, design or decision variables which can take discrete or continuous values or even a mixture of both. The function $f : \Re^n \to \Re$ is called the objective or the cost function; the functions $\phi_r : \Re^n \to \Re$, $(r = 1, 2, ..., m)$, are the (inequality) constraints, and the constants b_r, $(r = 1, 2, ..., m)$, are the bounds or limits of the constraints.

An optimization problem can generally be classed as either linear or non-linear depending on some characteristics taken by the objective and constraint functions. For instance, when the constraints $\phi_r(\boldsymbol{x})$, $(r = 1, 2, ..., m)$, take on linear form, the problem is regarded as a *linearly constrained* optimization problem. It is considered as a *linear programming problem*, if the constraints and the objective are all linear. Furthermore, if the decision variables take on integer values, the linear programming problem is known as *integer programming* or *integer linear programming*. If on the other hand, the objective is at most quadratic accompanied by linear constraints, the resultant optimization problem is regarded as *quadratic programming*.

Since we are able to define what an optimization problem is all about and briefly describe the different forms it can take; it is worthwhile to, also, briefly discuss what makes a given solution of an optimization problem to be regarded as optimal; knowing that the main goal behind the formulation of an optimization problem is to (if possible) find an optimal solution.

A given solution $\boldsymbol{x}^*$ is said to be optimal, provided that among all vectors that satisfy the constraints set, it has (respectively for minimization/maximization problem) the

smallest/largest objective value: for any s with $\phi_1(s) \le b_1, ..., \phi_m(s) \le b_m$ we have $f(s) \ge f(x^*)$. An optimum solution for different type of optimization problems with varying degree of complexity can be obtained whenever the objective is composed of either a univariate function or a multivariate function. When optimizing a single objective optimization problem, an optimum can either be its maximum or minimum depending upon what the problem is aimed at addressing. The following definitions are due to Weise [148].

6.2.1.1 Local Optimum

A local optimum of an optimization problem can either be a local minimum or a local maximum. Thus, a local optimum is said to be a local minimum [maximum], if for any $s \in X$ of single (objective) function $f : X \mapsto \Re$ it serves as an input element satisfying $f(s) \le f(x)\left[f(s) \ge f(x)\right]$ for all x in the neighbourhood of s.

Now, if $X \subseteq \Re$, we therefore can write:

$$\forall s \; \exists \varepsilon > 0 : f(s) \le f(x)\left[f(s) \ge f(x)\right] \forall x \in X, |x - s| < \varepsilon .$$

6.2.1.2 Global Optimum

A global optimum of an optimization problem can either be a global minimum or a global maximum. Thus, a global optimum $x^* \in X$ is said to be a global minimum [maximum], if for any $s \in X$ of single (objective) function $f : X \mapsto \Re$ it serves as an input element satisfying $f(s) \le f(x)\left[f(s) \ge f(x)\right] \forall x \in X$. The Figure below from Weise [148] shows some instances of both local and global optimum (minimum/maximum):

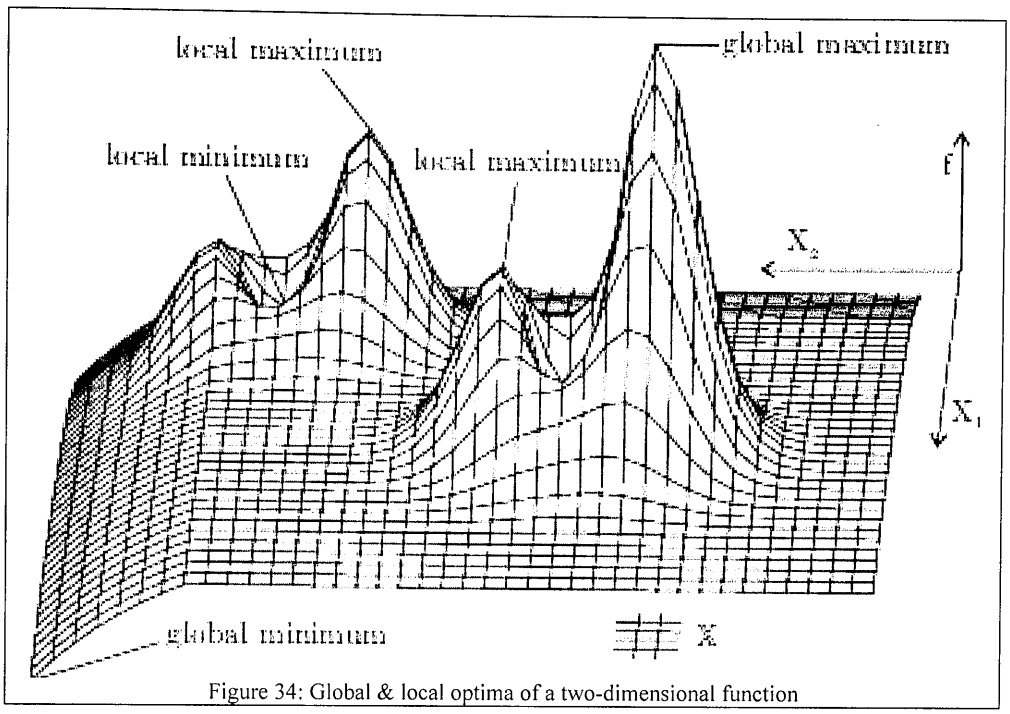


Figure 34: Global & local optima of a two-dimensional function

6.2.2 Standard Test Functions

There are numerous test functions in the literature that are used to test and evaluate the performance of optimization algorithms [155]. These tests can either be classed as either unconstrained or constrained problem optimization. An unconstrained optimization problem, as the name implies, is an optimization problem without any constraint attached to it; while the constrained one has some accompanying constraints that any candidate solution must satisfy for it to be feasible.

6.2.2.1 Unconstrained Optimization Problems

Whether an optimization problem is a constrained or unconstrained one, the main goal is to find an optimal solution that minimizes/maximizes the objective better than in any other feasible candidate solution in the solution space. In an unconstrained optimization, an optimal solution occurs at the critical points that makes the stationary condition $f'(x) = 0$ only if f is differentiable; However, this stationary condition is just

a necessary (but non-sufficient). It should be noted that, if $f'(x_*)=0$ and $f''(x_*)>0$, the solution x_*, is a local minimum; however, if $f'(x_*)=0$ and $f''(x_*)<0$, it is a local maximum. If on the contrary $f'(x_*)=0$, but $f''(x_*)$ is indefinite (both positive and negative) when $x \to x_*$ then x_* is a saddle point.

In testing our algorithms we decided to use some standard test functions that are widely reported in the literature [153, 154, 155]; these include De Jong's, Rastrigin's, Goldstein-Price's, Schewefel's and Beale functions.

(i) ***De Jong's function***: De Jong is a unimodal as well as a convex function with global optimum $f(\boldsymbol{x}_*)=0$ occurring at $\boldsymbol{x}_*=(0,0,...,0)$. The function is also known as a sphere function and is given by:

$$f(x)=\sum_{j=1}^{n} x_j^2, \quad -5.12 \le x_j \le 5.12 \qquad \forall j=1,...,n$$

The figure below shows a graphical representation of De Jong's function for $n = 2$:

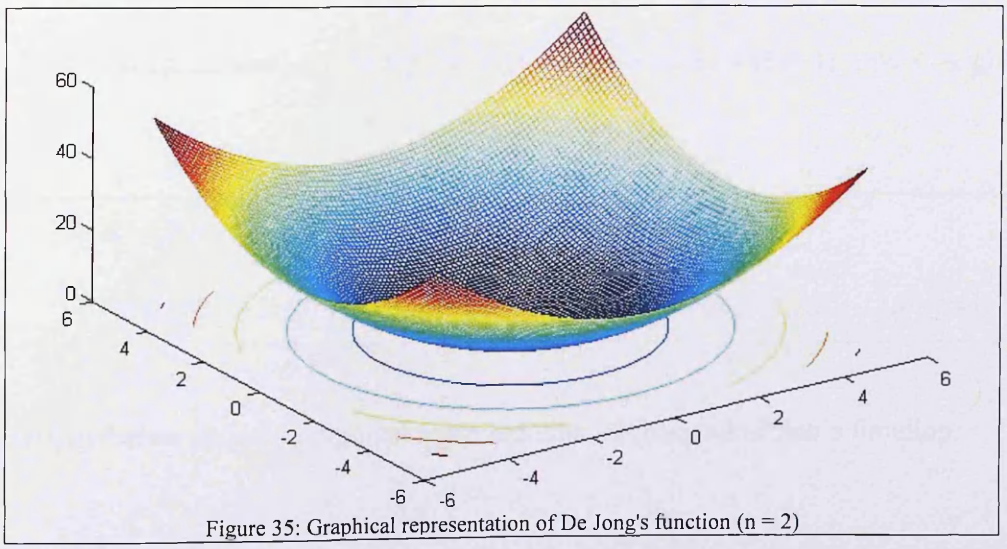


Figure 35: Graphical representation of De Jong's function (n = 2)

(ii) ***Rastrigin's function***: is a function whose terrain is contaminated by several local minima. Its global optimum $f(\boldsymbol{x}_*) = 0$ occurs at $\boldsymbol{x}_* = (0, 0, ..., 0)$; and the search domain is mostly restricted within $-5.12 \le x_j \le 5.12$. The function is given by:

$$f(x) = 10n + \sum_{j=1}^{n}\left(x_j^2 - 10\cos(2\pi x_j)\right), -5.12 \le x_j \le 5.12 \qquad \forall j = 1, ..., n$$

The figure below shows a graphical representation of Rastrigin's function for $n = 2$:

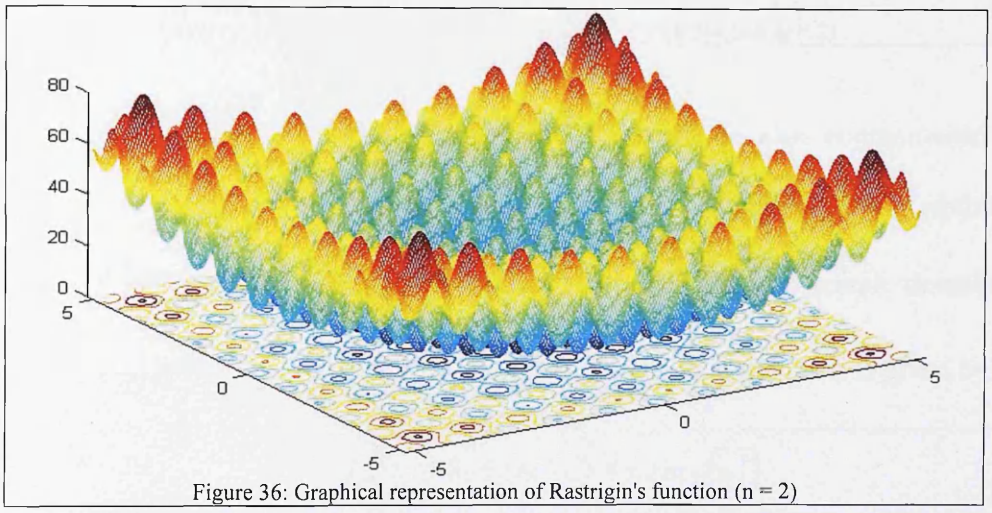


Figure 36: Graphical representation of Rastrigin's function (n = 2)

(iii) **Goldstein-Price's function**: is a little bit flat-terrain function with only two variables whose search domain is normally restricted within $-2 \le x_j \le 2$. The function has its global (minimum) solution $f(\boldsymbol{x}_*) = 3$ occurring at $\boldsymbol{x}_* = (0, -1)$; and it is given by:

$$f(x) = \left(1 + \left(x_j + x_{j+1} + 1\right)^2 \left(19 - 14x_j + 3x_j^2 - 14x_{j+1} + 6x_j x_{j+1} + 3x_{j+1}^2\right)\right)$$
$$* \left(30 + \left(2x_j - 3x_{j+1}\right)^2 \left(18 - 32x_j + 12x_j^2 + 48x_{j+1} - 36x_j x_{j+1} + 27x_{j+1}^2\right)\right) \qquad for\ j = 1$$

The figure below shows a graphical representation of Goldstein-Price's function:

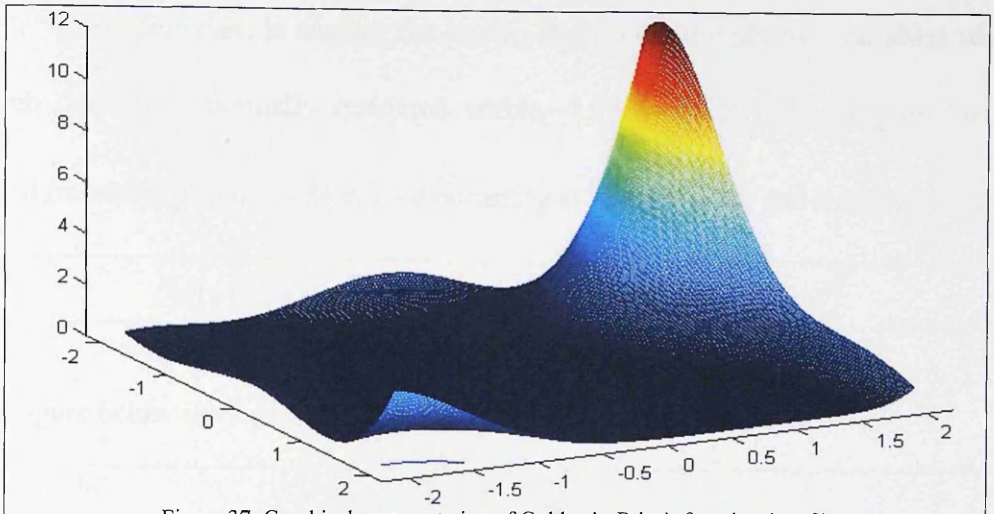


Figure 37: Graphical representation of Goldstein-Price's function (n = 2)

(iv) ***Schwefel's function***: is a function whose terrain is also contaminated by several local minima. Its global optimum $f(x_*) = 0$ is obtainable at $x_i = 420.9687\ (for\ i = 1,2,...,n)$; and the search domain is mostly restricted within $-500 \le x_i \le 500,\ for\ i = 1,2,...,n$. The function is given by:

$$f(x) = 418.9829n - \sum_{i=1}^{n}\left(x_i \sin\sqrt{|x_i|}\right)$$

The figure below shows a graphical representation of Schwefel's function for $n = 2$:

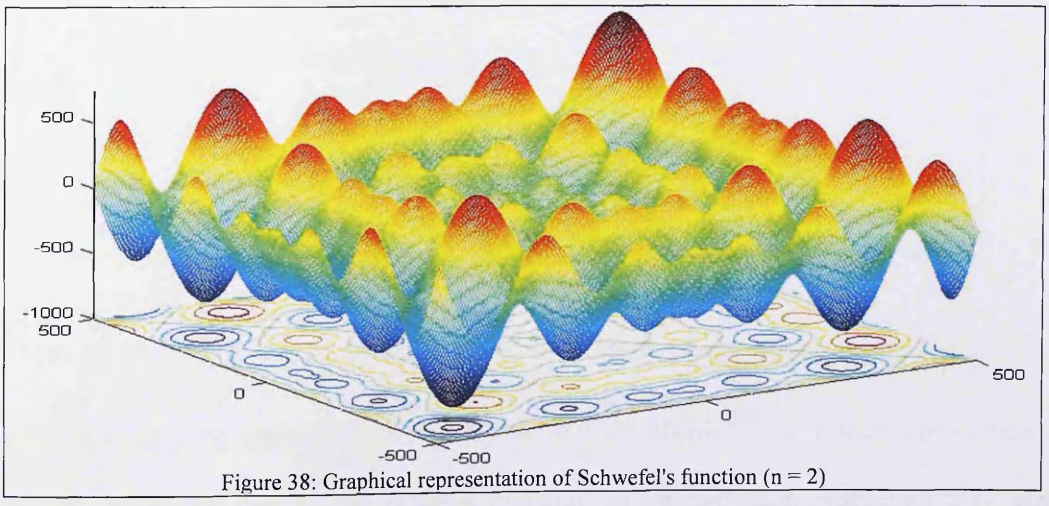


Figure 38: Graphical representation of Schwefel's function (n = 2)

(v) ***Beale function***: is another flat-terrain function with only two variables whose search domain is normally restricted within $-4.5 \le x_j \le 4.5$. The function has its global (minimum) solution $f(\boldsymbol{x}_*) = 0$ occurring at $\boldsymbol{x}_* = (3,\ 0.5)$; and it is given by:

$$f(x) = (1.5 - x_1 + x_1 x_2)^2 + (2.25 - x_1 + x_1 x_2^2)^2 + (2.625 - x_1 + x_1 x_2^3)^2$$

The figure below shows a graphical representation of Beale function:

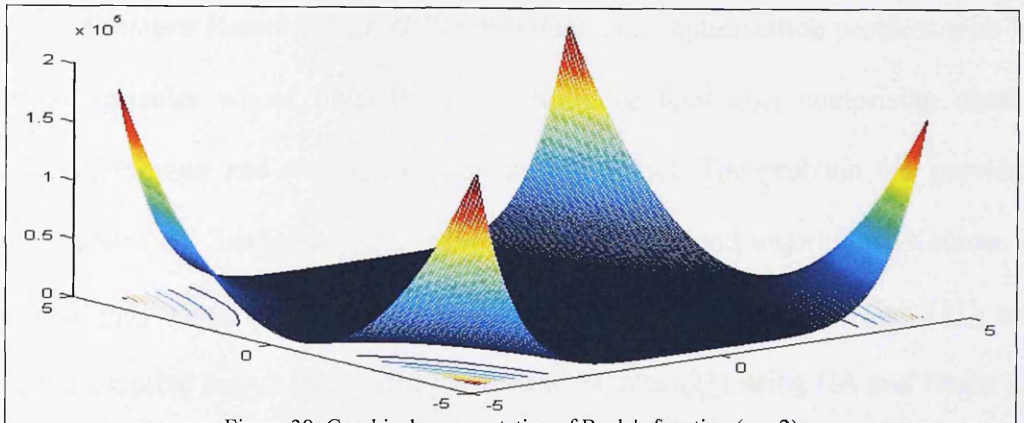


Figure 39: Graphical representation of Beale's function (n = 2)

6.2.2.2 Constrained Optimization Problems

A constrained optimization problem takes the form:

$$\begin{aligned} &\underset{x \in \Re^n}{\text{Minimize}}\ f(\boldsymbol{x}) \\ &\text{Subject to} \\ &\qquad \psi_r(\boldsymbol{x}) = 0, \quad (r = 1, 2, ..., m), \\ &\qquad \xi_s(\boldsymbol{x}) \le 0, \quad (s = 1, 2, ..., n), \\ &\qquad \boldsymbol{x} = (x_1, x_2, ..., x_n)' \in \Re^n \end{aligned}$$

Where f, ψ_r *and* ξ_s are real valued functions defined on the search space $S \subseteq R^n$.

In order to test the capability of our algorithms we decided to solve an optimization problem involving two of the popular constrained optimization problems: G11 and Pressure Vessel Design (*PVD*) problems as reported in Hedar [71] as well as in Global Optimization website [143]. The functions are defined as follows:

(i) ***G11 problem***: is a two-variable optimization problem whose global minimum $f(x^*) = \frac{3}{4}$ occurs at $x^* = \left(\pm\frac{1}{\sqrt{2}}, \frac{1}{2}\right)$. The problem is given by:

$$
\begin{aligned}
&Min\ f(x) = x_1^2 + (x_2 - 1)^2 \\
&subject\ to \\
&\qquad g(x): x_2 - x_1^2 = 0. \\
&\qquad -1 \le x_i \le 1, \quad i = 1,2
\end{aligned}
$$

(ii) ***Pressure Vessel Design (PVD) problem***: is an optimization problem with four design variables whose objective is to minimize total cost comprising costs of material, forming and welding of a cylindrical vessel. The problem has previously been tackled by Sandgren [128] using Branch and Bound algorithms; Kannan and Kramer [85] using an augmented Lagrangian Multiplier approach; Deb [31] using genetic adaptive search (GeneAS); Coello and Montes [23] using GA and Hedar [71] by using FSA. Coello and Montes [23], especially solved the problem using the following bounds $[1 \le x_1, x_2 \le 99, \ \& \ 10.0 \le x_3, x_4 \le 200.0]$; where x_1 & x_2 are considered as (real values rounded to the closest integer value) multiples of 0.0625; while x_3 & x_4 were considered as just floating point values.

The following figure shows the cylindrical vessel dissected to reveal the variables used in the optimization problem copied directly from Coello and Montes [23].

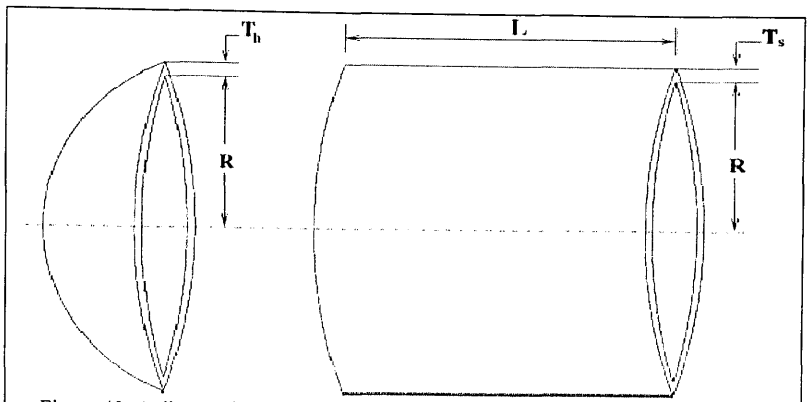


Figure 40: A dissected cylindrical vessel showing design variables in PVD problem

The 4 design variables are: x_1[T_s - thickness of the shell], x_2[T_h – thickness of the head], x_3[R - inner radius], and x_4[L - length of the vessel's cylindrical section excluding the head]. The minimization problem can be formulated by:

$$\begin{aligned}
&Min\ f(x) = 0.6224x_1x_3x_4 + 1.7781x_2x_3^2 + 3.1661x_1^2x_4 + 19.84x_1^2x_3 \\
&subject\ to \\
&\quad r(x): -x_1 + 0.0193x_3 \le 0, \\
&\quad s(x): -x_2 + 0.00954x_3 \le 0, \\
&\quad t(x): -\pi x_3^2 x_4 + \tfrac{4}{3}\pi x_3^3 + 1296000 \le 0, \\
&\quad u(x): x_4 - 240 \le 0,
\end{aligned}$$

6.2.3 Results and Discussions

In this part we aim to present the results obtained from our algorithms (PSO and SWAN) obtained by solving some standard optimization test functions (both constrained and unconstrained) in relation to those obtained by other studies conducted before. In all cases we conducted a total of 30 different trials in which we noted the best number of optimum solutions found in each case across the various algorithms.

For all problems, a given candidate solution is considered to be optimal if the absolute difference between it and the global optimum is less than a specified tolerance limit, ε. Mathematically, we can express this relation as follows:

$$If\left(\left|\textbf{\textit{global solution}} - \textbf{\textit{candidate solution}}\right| \le \varepsilon\right) Then\{$$
$$Terminate\ the\ Search\ Process$$
$$Return\ \textbf{\textit{candidate solution}}$$
$$\}$$

Where $\varepsilon = 10^{-10}$.

For the unconstrained solutions, we defined a success rate (%) as follows:

$$success\ rate = \frac{number\ of\ trials\ an\ optimal\ solution\ is\ returned}{Total\ number\ of\ trials} \times 100\%$$

The following is the result obtained after solving the unconstrained optimization problems

Problem	Number of Variables	Optimal Solution	Performance					
			Best		Worst		Success rate (%)	
			PSO	SWAN	PSO	SWAN	PSO	SWAN
De $Jong_2$	2	0.00	0.00	0.00	0.00	0.00	100.0	100.0
De $Jong_5$	5	0.00	0.00	0.00	0.00	0.00	90.00	100.0
De $Jong_{10}$	10	棚 00	0.00	0.00	0.00	0.00	76.67	90.00
$Rastrigin_2$	2	0.00	0.00	0.00	0.00	0.00	96.67	100.0
$Rastrigin_5$	5	0.00	0.00	0.00	0.00	0.00	73.33	83.33
$Rastrigin_{10}$	10	0.00	0.00	0.00	0.00	0.00	63.33	80.00
Goldstein-Price	2	3.00	3.00	3.00	3.00	3.00	93.33	100.0
Schwefel	2	0.00	0.00	0.00	1.2601	0.6445	30.00	53.33
Beale	2	0.00	0.00	0.00	0.00	0.00	96.67	100.0

Table 7: Comparison of the results for the unconstrained optimization problems

From Table 7 above, it can be observed that, our selected algorithms performed extremely well in returning quite a number of optimal solutions out of the total number of 30 trials conducted. For the De Jong's function we tried three different versions with 2, 5 and 10 variables respectively; similarly, for the Rastrigin's function, three versions were also implemented with 2, 5, and 10 variables respectively. The red coloured values, we believe, will draw the reader's attention: as to why the values for

the worst solutions generated by our algorithms do not (in anyway) differ from those of their corresponding best (optimal) in the same corresponding instances, despite the fact that in such cases it is apparent that the success rates are not up to 100%? The answer to this question can be answered when the reader is made to understand that, this is due to approximation. For instance, let us consider the Rastrigin function with 5 variables ($Rastrigin_5$) in which the PSO recorded a success rate of 73.33% (i.e. 22 out of the 30 trials conducted returned an optimal solution); the (actual) configuration of the worst solution generated by PSO in such instance is:

$\boldsymbol{x}$ = $\{8.70052\times10^{-7},\ 7.58468\times10^{-5},\ -3.21291\times10^{-7},\ 3.04434\times10^{-6},\ 6.51759\times10^{-6}\}$ in which the computed objective function value would be 1.1517×10^{-6} which is virtually equivalent to zero. Similarly, when we consider the Beale function with only two variables in which PSO recorded a success of 96.67% (i.e. 29 successful runs out of 30); the only worst solution has an objective value of $3.98985391337173\times10^{-6}$ which occurs at : $\boldsymbol{x} = \{2.998079, 0.4999075\}$.

The Schewefel's function optimization appears to be extremely difficult for both algorithms (PSO and SWAN) going by the success rates they recorded. PSO recorded only 30% success (9 successful out of the total 30 runs); while SWAN managed to record a little bit more than half of the total number of runs with 53.33% success (16 successful out of the total of 30). This difficulty, we believe, is as a result of several local optimum solutions that characterizes the search space of the Schwefel's function coupled with a relatively vast search domain: $-500 \le x_i \le 500,\ for\ i = 1,2,...,n$.

In the constrained problems, we compared our results with those obtained in some previous studies, especially those of Filter Simulated Annealing (FSA) reported in

Hedar [71]. The following table shows the comparison of results obtained for the G11 constrained case.

Problem	Optimal Solution	Degree of Performance	This research		FSA [71]
			PSO	SWAN	
G11	$f(x^*) = \frac{3}{4}$ $x^* = \left(\pm\frac{1}{\sqrt{2}}, \frac{1}{2}\right)$	Best	0.750000 (7.56)	0.750000 (11.12)	0.749999
		Average	0.753377 (7.01)	0.750132 (9.91)	0.749999
		Worst	0.760192 (10.11)	0.750823 (13.21)	0.749999

Table 8: Comparison of the results for the G11 constrained problem

Although we are unable to reproduce the run times for the FSA in Hedar [71]; Table 8 displays (in parentheses) among other things the (best, average and worst) run times (in secs) for PSO and SWAN.

From Table 8 above, it can be seen that, although our algorithms' average performances are inferior to those obtained by FSA as reported in Hedar [71]; still they were able to obtain good solutions in quite a number of runs. The average solutions for the PSO and SWAN are respectively: $\{x_1, x_2\}$ = {0.707664, 0.497418} and {0.707059, 0.499800}. The SWAN also seems to have an upper hand over PSO in this instance in which its worst solution (0.750823) occurring at $\{x_1, x_2\}$ = {0.707664, 0.497418} seems slightly better than the worst solution (0.760192) returned by PSO with $\{x_1, x_2\}$ = {0.707107, 0.489910}.

The Table 9 below compares the results obtained by our designed algorithms to other studies conducted previously and reported in the literature concerning the *PVD* problem:

Design Variables and constraints	Best solution found						
	This Research		FSA [71]	GA [23]	GeneAS [31]	Kannan [85]	Sandgren [128]
	PSO	SWAN					
$x_1[T_s]$	0.7900814	0.7890814	0.76832571	0.812500	0.937500	1.12500	1.12500
$x_2[T_h]$	0.3900000	0.3890010	0.37978380	0.437500	0.500000	0.62500	0.62500
$x_3[R]$	40.549900	40.545510	39.8096222	42.097398	48.32900	58.29100	47.7000
$x_4[L]$	197.01002	196.90000	207.225559	176.65405	112.6790	43.69000	117.7010
r(x)	-0.007468	-0.0065531	-9.9087E-13	-0.000020	-0.00475	**0.000016**	-0.20439
s(x)	-0.003154	-0.0021968	-5.4296E-11	-0.035891	-0.038941	-0.068904	-0.16994
t(x)	-986.9696	-107.69857	-10.706469	-27.88607	-3652.8768	-21.22010	**54.22601**
u(x)	-42.98998	-43.100000	-32.774405	-63.34595	-127.32100	-196.3100	-122.299
Best f(x)	5960.2483	5946.9668	**5868.76484**	6059.9463	6410.3811	7198.0428	8129.104
Average f(x)	6199.4917	**6065.1485**	6164.58587	6177.2533	N/A	N/A	N/A
Worst f(x)	6762.0298	**6456.6611**	6804.32810	6469.3220	N/A	N/A	N/A
Std dev	350.19279	207.51562	257.473670	130.92970	N/A	N/A	N/A

Table 9: Comparison of the results for the *PVD* problem

From Table 9 above, the reader can easily notice how our algorithms performed well in relation to the previous studies conducted on the *PVD* problem. Although, FSA by Hedar [71], to our knowledge, is the best known solution with a global minimum of $f(x^*) = 5868.764836$ occurring at $x^* = \{0.768325709391, 0.379783796302, 39.809622248187, 207.225559518596\}$; however, we feel there are other positive things to consider about our algorithms too. For instance, it can easily be seen that, our algorithms performed better than all other previous algorithms except for the Hedar's FSA. SWAN produced the next best solution, and followed closely by PSO; while Coello and Montes' GA occupies the 4th position. It can also be observed that, the average solution (6065.1485) produced by SWAN was better than any of the average solutions produced by any other technique shown on Table 9 above.

Furthermore, it could be seen that even the worst solution [6456.6611 whose variables take on values: $\{x_1, x_2, x_3, x_4\} = \{0.798001, 0.513715, 39.9022, 206.012487\}$] generated by our SWAN algorithm outperformed all the other worst solutions recorded in the previous researches as presented on the table above. Additionally, even the PSO's worst solution [(6762.0298) with variables configuration: $\{x_1, x_2, x_3, x_4\} =$

{0.851091, 0.493715, 40.81002, 197.12400}] is better than the FSA's worst solution with objective value of 6804.32810. Now, considering the degree of variability of solution generation; it can be observed that, there is an indication that, out of the total number of trials conducted; SWAN produced more good solutions than FSA as evidenced by the value of its standard deviation (207.51562) as against the FSA's 257.473670. This simply means, probably SWAN slightly misses the chance of producing the best solution.

One other thing to note especially from the table is that, all the algorithms (PSO, SWAN, FSA, GA, and GeneAS) which produce very good, but still feasible solutions allocate larger values to design variable (x_4 – length of the vessel's cylindrical section excluding the head); implying that varying other design variables while allocating larger values to x_4 is the most reasonable decision for producing feasible vessel's design at lower cost. Note that Kannan and Kramer's [85] approach is the only algorithm that allocates lower value (43.690000) to variable x_4; and consequently, ended up with an infeasible solution by slightly violating the first constraint whose value is 0.000016. The Branch and Bound algorithm of Sandgren [128], however, was the worst among all those reported in Table 9; in the sense that, apart from producing an infeasible solution (due to the violation of the 3rd constraint), it also provided a solution with the worst objective value of 8129.1036.

6.3 Test of Statistical Hypothesis

The primary objective of this section is to compare the average performance of SWAN and FSA algorithms in relation to the optimization of the *PVD* problem introduced in the previous section, using a well known statistical tool, namely, the test of statistical

hypothesis (TSH). The TSH would be used for assessing and comparing the effectiveness of both algorithms in obtaining a high quality solution, when the search begins from diverse and randomly generated initial solutions. In this context, we found the definition given by Hassan *et al* [69] as most appropriate; who defined Effectiveness as: "*... the ability of a given algorithm to, repeatedly, find a global [best known] solution or arrives at sufficiently close solution when the algorithm is [re]started from many [different] random points [initial solutions] in the design space*". In other words, Effectiveness can be defined as the probability of obtaining (on average) a high quality solution.

It should be understood that, effectiveness, in our research, relates to how an algorithm is (on average) able to produce high quality solutions that are close to the global or best known solution. However, before we descend into the detailed analysis of algorithmic performances; it is important and worthwhile to have a brief look at the elements of TSH.

Definition: A statistical hypothesis is a statement concerning the probability distribution of a random variable or population parameters that are inherent in a probability distribution [122].

In any hypothesis testing problem, null (H_0) and alternative (H_1 or H_a) hypotheses are formulated, such that when the H_0 is rejected, the H_1 has to be accepted and vice versa. The H_0 is usually a statement made such that when the objective of an experiment is to establish a claim, the nullification of the claim should be taken as the H_0. The experiment is often performed to determine whether the H_0 is false. For instance,

consider a situation when a prosecution wants to establish (with evidence) that a certain person is guilty of a crime. The H_0 in this case, should be that the person is innocent; while the H_1 would be that the person is guilty; thus, the claim now becomes the alternative. It should be noted that, decisions are always drawn with respect to the H_0; in the sense that failure to reject it does not necessarily means it (H_0) is true. For example, when a person is judged "not guilty" does not necessarily means he/she is innocent, but rather the prosecution fails to provide evidence (beyond reasonable doubt) against him/her to nullify the presumption of innocence.

In TSH, there is a chance (probability) that H_0 will be rejected, when in fact it should not have been rejected (i.e. H_0 = True); and this probability is known as type I error. On the other hand, a type II error is when H_0 is accepted when in fact it is false (i.e. H_0 = False) and should be rejected. The table below shows the possible decision options available in any TSH.

	Actual situation of H_0	
Decision Taken	H_0 is True	H_0 is False
Do not Reject H_0	Correct Decision	Type II error (β)
Reject H_0	Type I error (α)	Correct Decision

Table 10: Possible decisions in a test of statistical hypothesis (TSH)

There are basically five elements (steps) involved in conducting a TSH [122, 144]; and these include the following:

1. The H_0 and H_1 are formulated, defined and stated; where H_0 is usually the nullification of a claim, while H_1 is normally the claim itself.
2. Decide on the desired level of significance α, which is an important parameter in deciding the critical/rejection region of the test under consideration.

3. Determine and compute the value of the appropriate test statistic; which is a function of the sample measurements upon which the decision of rejection/non rejection of H_0 will be based.

4. Determine the rejection/critical region of the test – a region (depending on the size of the level of significance) specifying the values of the observed test statistic for which H_0 will be rejected.

5. Draw a conclusion: if the value of the (observed) test statistic (computed from step 3 above) falls in the rejection region (defined in step 4), then H_0 has to be rejected; and one concludes that there is enough evidence [from the sample] to decide in favor of the alternative H_1, otherwise the conclusion will be that: H_0 cannot be rejected based on the premise of non sufficient evidence from the sample.

Now coming back to the discussion of our main goal in this section; which is to answer a research question in relation to the constrained *PVD* problem discussed in section 6.2 above; and this is to test (by using techniques of TSH) whether (based on the sample information available):

The average performance of SWAN algorithm is better than that of FSA.

At this point we want the reader to be aware that, the word '*better*' in this context means having smaller objective value (since *PVD* is a cost minimization problem); so if we have two or more algorithms in which one among them produces feasible solutions with lower objective value than the rest; then it should be adjudged as *better* than them.

Now using mathematical notation, we want to represent average performances of SWAN and FSA by ${}^{SWAN}\mu_{perf}$ and ${}^{FSA}\mu_{perf}$ respectively. In order to test our hypothesis, we are going to sequentially follow the steps outlined above as follows:

*STEP*1:

$H_0 : {}^{SWAN}\mu_{perf} \geq {}^{FSA}\mu_{perf}$

$H_1 : {}^{SWAN}\mu_{perf} < {}^{FSA}\mu_{perf}$

*STEP*2:

(*i*) $\alpha = 5\%$ *and* (*ii*) $\alpha = 1\%$

*STEP*3:

$$\text{Test statistic, } Z = \frac{{}^{SWAN}\bar{x}_{perf} - {}^{FSA}\bar{x}_{perf}}{\sqrt{\frac{\sigma^2_{SWAN}}{n_{SWAN}} + \frac{\sigma^2_{FSA}}{n_{FSA}}}};$$

where:

${}^{SWAN}\bar{x}_{perf}$ *is the sample average performance of SWAN and is equal to* 6065.1485,

${}^{FSA}\bar{x}_{perf}$ *is the sample average performance of FSA and is equal to* 6164.5859,

$s^2_{SWAN}\left(\text{estimate of } \sigma^2_{SWAN}\right)$ *is the sample variance for SWAN and is equal to* $(207.5156)^2$,

$s^2_{FSA}\left(\text{estimate of } \sigma^2_{FSA}\right)$ *is the sample variance for FSA and is equal to* $(257.4737)^2$,

n_{SWAN} *is the sample size (number of trials) for SWAN and is equal to* 30,

n_{FSA} *is the sample size (number of trials) for FSA and is equal to* 30.

Thus;
$$Z = \frac{6065.1485 - 6164.5859}{\sqrt{\frac{(207.5156)^2}{30} + \frac{(257.4737)^2}{30}}} = -1.6469$$

*STEP*4:

The rejection region for

(*i*) $\alpha = 5\%$ *is*: *reject* H_0 *if* $Z < -Z_\alpha$; *where* $-Z_{0.05} = -1.6450$.

Therefore, since $Z < -Z_{0.05}$ $(i.e. -1.6469 < -1.6450)$; *we then* ***reject*** $\boldsymbol{H_0}$

(*ii*) $\alpha = 1\%$ *is*: *reject* H_0 *if* $Z < -Z_\alpha$; *where* $-Z_{0.01} = -2.3260$.

Thus, since $Z > -Z_{0.01}$ $(i.e. -1.6469 > -2.3260)$; *we then* ***Do not reject*** $\boldsymbol{H_0}$

*STEP*5:

(*i*) Since the decision reached in step4 is to ***reject*** $\boldsymbol{H_0}$, we conclude that: The random sample ***provides*** sufficient evidence to believe that, the average performance of SWAN is better than that of FSA at 5% level of signinficance.

(*ii*) Since the decision reached in step4 is ***not to reject*** $\boldsymbol{H_0}$, we conclude that: The random sample ***do not provides*** sufficient evidence to believe that, the average performance of SWAN is better than that of FSA at 1% level of significance.

Table 11: A TSH analysis between SWAN and FSA

From Table 11 above, it can be observed that, we tested for the validity of the claim that: "*The average performance of SWAN is better than that of FSA*" under two

different levels of significance (i.e. $\alpha = 5\%$ and $\alpha = 1\%$). It can be seen that, for 5% level of significance, the sample provided [just] enough evidence to reject the H_0, thereby warranting the acceptance of H_1; and this literally means that: we are 95% confident that: *the average performance of SWAN is indeed better than that of FSA.* However, for 1% level of significance, the sample **does not** provide enough evidence to reject H_0, thereby warranting its acceptance; and this literally means that: at 1% level of significance, we cannot claim that: *the average performance of SWAN is better than that of FSA.*

7.0 Conclusion and Future Research

7.1 Conclusion & Future Work

In this thesis we addressed the classical (unconstrained) as well as the constrained PSP using some selected metaheuristic algorithms that are widely reported and implemented in the literature. PSP being one of the most widely studied areas in finance has drawn much interest from both academia and industry enjoying varying implementation strategies using approximate (especially metaheuristic) algorithms. In this research too, like in several others before it; we solved the classical problem using six different metaheuristic search techniques of which three (SA, TS, and Parallel SA) among them are Local searches, two (GA and PSO) are EAs and the last one (SWAN) is a hybrid of SA and PSO.

7.1.1 Conclusion

The entire thesis report has been partitioned into seven different chapters, each with its subsections aimed at giving thorough details of what has been discussed therein. Chapter one, for instance, discussed in great detail the origin of the PSP, what the classical Markowitz *E-V* model is all about, and various critics' view on some shortcomings that came along with the model. In the same chapter we discussed why this research is important as well as what makes it difficult and challenging. The major contributions that this research has on offer were outlined in chapter one and these, once again, include:

- Designing and implementing an algorithm (entitled SWarm ANnealing – SWAN) which a hybrid of PSO and SA aimed at exploiting the PSO's exploration (diversification) potential as well as SA's exploitation

(intensification) ability to return very promising solutions for both constrained and unconstrained cases of PSP.

- Designing and implementation of a neighbourhood structure (entitled IDDIT) for Local searches aimed at guiding our algorithms to explore the neighbourhood of the incumbent solutions in order to either find a very promising solution or escape entrapment in local optima.
- Developing and implementation of (even more challenging and more advanced) neighbourhood structure purposely designed for our swarm algorithms (PSO and SWAN). This neighbourhood definition strategy has some form of guidance enabling the algorithms to intelligently decide which asset should be deleted from or inserted into a candidate portfolio.
- Proposing a model (based on the ideas put forward by a previous research) for solving PSP with a semivariance as an alternative to the conventional objective (variance); while at the same time incorporating some real world (cardinality and floor & ceiling) constraints.

Chapter two built on the description of the classical Markowitz E-V model provided in the chapter one, and further defined what constitutes an E-V investor. Limitations and shortcomings of the classical E-V model were outlined therein and various implementations (in the literature) to address the problems that such shortcomings pose were studied. Chapter two also discusses the possible strategies of extending the classical E-V model which can be achieved by either substituting the original objective (the variance) with other alternatives (such as mean absolute deviation, semivariance, value at risk, e.t.c.) or incorporating some other real world constraints (such as cardinality, floor & ceiling, transaction cost, e.t.c.) and sometimes both. The chapter

concluded by describing the conventional computation of semivariance as well as a thorough explanation about the implementation of our proposed mean-semivariance portfolio selection model.

In chapter three, we provided some acceptable definitions of metaheuristics as provided in the literature. We also reviewed some literature on the successful applications and implementations of metaheuristic algorithms in other diverse research areas as reported in the literature. Due to the fact that, classical PSP itself can be viewed as a bi-objective, and indeed a variant of multiobjective optimization problems; we decided to review other applications and successful implementation of heuristics/metaheuristics in other multiobjective optimization areas of research. In the same chapter also, we specifically decided to look at some other related researches that deals with the metaheuristic applications to portfolio selection. In the final part, the chapter provided a thorough explanation of the six metaheuristic algorithms used in this research.

The algorithmic details concerning the practical implementation of the classical (unconstrained) PSP were described in chapter four. These details range from the different decisions related to parameter choice to the thorough description of how the bi-objective PSP problem is implemented. The bi-objective problem is such that we aim to select a portfolio of assets which minimizes the portfolio risk while at the same time maximizing the portfolio return such that its gap to the supplied target return is made as small and as negligible as possible. The chapter also provides an explanation to some set of popular performance and evaluation metrics which would be used to assess the performance of our algorithms amongst themselves as well as against a well

known non-linear optimization solver (The CPLEX). The chapter concludes with discussion of the results obtained for the solutions of the classical PSP from the solver and our implemented algorithms.

The fifth chapter was dedicated to discussing the details involved in the implementation of the constrained PSP. Because of the challenges involved in dealing with a constrained formulation; we decided to give a thorough explanation on how we represent a candidate solution – which involves two different sets; the integer set of assets' indices that constitutes a candidate portfolio and a set of real numbers representing the actual portfolio funds invested in the corresponding elements (indices) in the integer set. The chapter went ahead to provide a detailed explanation of the two neighbourhood move definition strategies developed in this research (which form integral parts of this research's contribution). The first one (entitled IDDIT) is meant to serve our Local search algorithms; while the other (which is more advanced having IDDIT-like operations and some forms of guidance) is meant to serve our swarm algorithms. The chapter also discusses the type of repair mechanism implemented which is aimed at enforcing feasibility and constraints satisfaction. The chapter concludes by discussing the results and evaluation of the algorithms.

Chapter six is meant to assess how well our algorithms (PSO and SWAN) perform in solving some other applications of optimization problems other than the PSP they were originally designed for. The chapter gave a very brief overview of some popular, but standard optimization test functions that are used to evaluate newly developed optimization algorithms. The search space's terrain of most of these test functions are characterized by multiple local solutions which makes it quite challenging for any

algorithms to easily locate the global solution. We decided to look at solutions of minimization problems for both constrained (involving *PVD* and G11 constrained problems) and unconstrained instances (involving De Jong's, Rastrigin, Goldstein-Price, Shwefel's and Beale test functions); after which we compare our results with some previous ones found in the literature. Going by the degree of success recorded by one of our algorithms (SWAN) in solving the *PVD* problem; we decided to answer a certain hypothetical question we formulated using one of the major statistical tools (statistical hypothesis testing) which can be used to establish the validity or otherwise of the hypothesis. The hypothetical statement we wanted to assess its validity was: "*The average performance of SWAN algorithm is better than that of FSA*". The hypothetical statement was tested under two different levels of significance (5% and 1%) after which appropriate conclusions were drawn.

7.1.2 Future works

There are several ways in which any given research can be taken to the next level; in our own particular case, our future plan will center on dedicating much of our available time to further our research on metaheuristic algorithms designed to solve difficult optimization problems. We will be much interested in the technicalities that explains why a given algorithm behaves the way it does; this has to do with various algorithmic parameter settings and their effect on performance. We plan to, initially, begin with GA by thoroughly analyzing its various parameter settings (involving crossover and mutation probabilities, choice of suitable number of chromosomes, elitism issues and many more). This drive is motivated by the relatively poor performance of the GA in solving (even the relatively simpler) unconstrained formulation of the PSP; we plan to give the algorithm another chance by thoroughly

looking at the above mentioned factors in relation to how they were implemented to solve the problem at hand in order to see if the algorithm would live to its expectations as one of the best performing EAs.

SWAN has, undoubtedly, proved itself as a very promising (hybrid) algorithm, whose success can be credited to the diversification power of PSO coupled with the intensification potential of SA. The algorithm was found to be very competitive, while at the same time excelling in producing very good results, irrespective of whether it is PSP or other applications of optimization problems. As part of our future work; in order to fully understand the behaviour of the (SWAN) algorithm and to explore more of its capabilities, we plan to implement it in such a way that the SA part comes before the PSO part, and this essentially means to start with the SA processes until convergence, after which the best solution found will be used to (further) generate some (*particles*) candidate solution which would serve as the key players in the subsequent PSO processes; by doing so, we hope to come up with even stronger algorithm having better search abilities.

We also plan to raise the difficulty level of our proposed PSP model by incorporating additional real world constraints (such as class/sector constraints, transaction costs, roundlot constraints and even integer variables formulation); so that the model will be as representative of investors' real world decision options as possible; and these would consequently leads to the development of even more advanced and intelligent neighbourhood structures for both our local searches and swarm algorithms.

REFERENCES

1. Aarts, E.H.L and Korst, J. (1989). Simulated Annealing and Boltzman Machines: A Stochastic Approach to Combinatorial Optimization and Neural Computing. *John Wiley & Sons,* Chichester, UK.

2. Abido, M. A. (2009). Multiobjective particle swarm optimization for environmental/economic dispatch problem. *Electric Power Systems Research*, 79(7): 1105 – 1113.

3. Alaya, I., Solnon, C. and Ghedira, K (2007). Ant Colony Optimization for Multi-objective Optimization Problems. In *19th IEEE International Conference on Tools with Artificial Intelligence (ICTAI 2007)*, 1: 450 – 457.

4. Armananzas, R. and Lozano, J. A. (2005). A Multiobjective Approach to the portfolio Optimization Problem. *In The 2005 IEEE Congress on Evolutionary Computation*, 1388 – 1395.

5. Arora, S. and Barak, B. (2009). Computational Complexity: A Modern Approach. *Cambridge University Press.*

6. Atkinson, A. B. (1970). On the measurement of inequality. *Journal of Economic Theory*, 2, 244 – 263.

7. Ballestero, E. (2005). Mean-Semivariance Efficient Frontier: A Downside Risk Model for Portfolio Selection. *Applied Mathematical Finance*, 12(1): 1 – 15.

8. Baykasoglu, A. (2001). Goal Programming Using Multiple Objective Tabu Search. *The Journal of Operational Research Society*, 52(12), 1359 – 1369.

9. Bhandarkar, S. M. and Chirravuri, S. (1996). A study of Massively Parallel Simulated Annealing Algorithms for chromosome reconstruction via clone ordering. *International Journal of Parallel, Emergent and Distributed Systems*, 9(1&2): 67–89.

10. Bienstock, D. (1996). Computational Study of a Family of Mixed – Integer Quadratic Programming Problems. *Mathematical Programming*, 74: 121 – 140.

11. Blum, C. and Roli, A. (2003). Metaheuristics in Combinatorial Optimization: Overview and Conceptual Comparison. *ACM Computing Surveys*, 35(3): 268 – 308.

12. Boyd, R. and Richerson, P. J. (1985). Culture and the evolutionary process. *University of Chicago Press*.

13. Boyd, S. P. and Vandenberghe, L. (2004). Convex Optimization. *Cambridge University Press*, New York, USA.

14. Briant, O., Naddef, D. and Mounie, G. (2008). Greedy approach and multi-criteria simulated annealing for the car sequencing problem. *European Journal of Operational Research*, 191: 993 – 1003.

15. Burke, E. K. and Kendall, G. (2005). Introduction. *In SEARCH METHODOLOGIES: Introductory Tutorials in Optimization and Decision Support Techniques*. Chapter 1. E. K. Burke, and G. Kendall (editors). Springer Science + Business Media Inc., NY.

16. Calvet, L. E., Campbell, J. Y. and Sodini, P. (2009). Fight or Flight? Portfolio Rebalancing by Individual Investors. *The Quarterly Journal of Economics*, 124(1): 301 – 348.

17. Carlisle, A. and Dozier, G. (2001). An off-the-shelf PSO. In *Proceedings of the Workshop on Particle Swarm Optimization*, Indianapolis, USA.

18. Catanas, F. (1998). On a neighbourhood structure for portfolio selection problems. *Technical Report*, Departamento de Metodos Quantitativos do ISCTE, Lisboa, Portugal.

19. Cerny, V. (1985). Thermodynamical approach to the Travelling Salesman Problem: An Efficient Simulation Algorithm. *Journal of Optimization Theory and Applications*, 45(1), 41 – 51.

20. Chang, -T. J., Meade, N., Beasley, J. E. and Sharaiha, Y. M. (2000). Heuristics for cardinality constrained portfolio optimization. *Computers & Operations Research*, 27: 1271 – 1302.

21. Chen, W., Zhang, R-T., Cai, Y-M. and Xu, S. (2006). Particle Swarm Optimization for Constrained Portfolio Selection Problems. In *Proceedings of the Fifth International Conference on Machine learning and Cybernetics.*

22. Clash, J. M. (1999). Focus on the Downside. *Forbes* (02.22.99); available at: www.forbes.com/forbes/1999/0222/6304162a.html.

23. Coello, C. A. C. and Montes, E. M. (2002). Constraint-handling in genetic algorithms through the use of dominance-based tournament selection. *Advanced Engineering Informatics*, 16, 193 – 203.

24. Coley, D. A. (1999). An Introduction to Genetic Algorithms for Scientists and Engineers. *World Scientific Publishing Company*, Singapore.

25. Crama, Y. and Schyns, M. (2003). Simulated annealing for complex portfolio selection problems. *European Journal of Operational Research*, 150: 546 – 571.

26. Cura, T. (2009). Particle swarm optimization approach to portfolio optimization. *Nonlinear Analysis: Real World Applications*, 10: 2396 – 2406.

27. Cvijovic, D. and Klinowski, J. (1995). Taboo Search: An Approach to the Multiple Minima Problem. *Science*, 267(5198): 664 – 666.

28. Da Silva, A. P. A. and Falcao, D. M. (2008). Fundamentals of Genetic Algorithms. In *Modern Heuristic Optimization Techniques: Theory and Applications to Power Systems*, K. Y. Lee and Mohamed A. El-Sharkawi (editors), A *John Wiley & Sons*, Inc., Hoboken, New Jersey, USA.

29. Darwin, C. R. (1859). On the Origin of Species by Means of Natural Selection or the Preservation of Favoured Races in the Struggle for life. *John Murray*, London.

30. de Werra, D. and Hertz, A. (1989). Tabu Search Techniques: A tutorial and applications to neural networks. *OR Spectrum*, 11, 131 – 141.

31. Deb, K. (1997). GeneAS: a robust optimal design technique for mechanical component design. In (Dasgupta. D, Michalewicz. Z. editors) Evolutionary algorithms in engineering applications. *Springer*, Berlin.

32. Deb, K. (2001). Multiobjective optimization using evolutionary algorithms. *John Wiley and Sons*, Chichester, UK.

33. Deb, K., Pratab, A., Agarwal, S. and MeyArivan, T. (2002). A fast and elitist multiobjective genetic algorithm: NSGA-II. *IEEE Transactions on Evolutionary Computation*, 6(2), 182 – 197.

34. di Tollo, G. and Roli, A. (2006). Metaheuristics for the Portfolio Selection Problem. *Technical Report R-2006-005*, Dipartimento di Scienze, Universita "G. D'Annunzio" Chieti-Pescara.

35. Donohue, C. and Yip, K. (2003). Optimal Portfolio Rebalancing with Transaction Costs. *The Journal of Portfolio Management*, 29(4): 49 – 63.

36. Dorigo, M. and Di Caro, G. (1999). Ant colony optimization: a new meta-heuristic. In *Proceedings of the 1999 IEEE Congress on Evolutionary Computation*, vol. 2, 1470 – 1477.

37. Dowsland, K. A. (1995). Simulated Annealing. In *Modern Heuristic Techniques for Combinatorial Problems*, C. R. Reeves (editor). *McGraw-Hill International* (UK) Limited.

38. Dowsland, K. A. (1995). Variants of Simulated Annealing for Practical Problem Solving. In *Application of Modern Heuristic Methods*, V. J. Rayward-Smith (editor), Alfred Waller Limited, Publishers, Oxon.

39. Dreo, J., Petrowski, A., Siarry, P. and Taillard, E. (2006). Metaheuristics for Hard Optimization Methods and Case Studies. *Springer-Verlag*, Berlin Heidelberg.

40. Dueck, G. and Winker, P. (1992). New concepts and algorithms for portfolio choice. *Applied Stochastic Models and Data Analysis*, 8, 159 – 178.

41. Eberhart, R. C. and Hu, X. (1999). Human Tremor Analysis Using Particle Swarm Optimization. In *Proceedings of the Congress on Evolutionary Computation*, 1927-1930.

42. Eberhart, R. C. and Shi, Y. H. (1998). Comparison between genetic algorithms and particle swarm optimization. 7th *Annual Conference on Evolutionary Programming*, 1447: 611 – 616.

43. Eglese, R. W. (1990). Simulated Annealing: A tool for Operational Research. *European Journal of Operational Research*, 46: 271 – 281.

44. Erera, A., Karacik, B. and Savelsbergh, M. (2008). A dynamic driver management scheme for less-than-truckload carriers. *Computers and Operations Research,* 35: 3397 – 3411.

45. Estrada, J. (2000). The Cost of Equity in Emerging Markets: A Downside Risk Approach. *Emerging Markets Quarterly* (Fall 2000). 19 – 30.

46. Estrada, J. (2002). Systematic Risk in Emerging Market: The D-CAPM. *Emerging Markets Review*, 3(4): 365 – 379.

47. Estrada, J. (2006). Downside Risk in Practice. *Journal of Applied Corporate Finance,* 18(1) 117 – 125.

48. Estrada, J. (2007). Mean-Semivariance Behaviour: Downside Risk and Capital Asset Pricing. *International Review of Economics and Finance*, 16(2): 169 – 185.

49. Estrada, J. (2008). Mean-Semivariance Optimization: A Heuristic Approach. *Journal of Applied Finance – Spring/Summer 2008*: 57 – 72.

50. Fabozzi, F. J., Kolm, P. N., Pachamanova, D. A. and Focardi, S. M. (2007). Robust Portfolio Optimization and Management. *John Wiley & Sons, Inc., Hoboken, New Jersey.*

51. Fama, E. F. (1970). Efficient Capital Markets: A Review of Theory and Empirical Work. *The Journal of Finance*, 25(2): 383 – 417.

52. Farrell, J.L., (1997). PORTFOLIO MANAGEMENT: Theory & Applications, 2nd ed. *Mc-Graw-Hill International Editions*, Finance Series.

53. Feiring, B. R., Wong, W., Poon, M. and Chan, Y. C. (1994). Portfolio Selection in downside risk optimization approach: application to the Hong Kong stock market. *International Journal of Systems Science*, 25(11): 1921 – 1929.

54. Feng, Y., Zheng, B. and Li, Z. (2010). Exploratory study of sorting particle swarm optimizer for multiobjective design optimization. *Mathematical and Computer Modelling*, 52(11–12), 1966 – 1975.

55. Fernandez, A. and Gomez, S. (2007). Portfolio selection using neural networks. *Computers & Operations Research*, 34: 1177 – 1191.

56. Fishburn, P. C. (1977). Mean-Risk Analysis with Risk Associated with Below-Target Returns. *American Economic Review*, 67, 116 – 126.

57. Gallego, R. A., Alves, A. B., Monticelli, A. and Romero, R. (1997). Parallel Simulated Annealing applied to long term transmission network expansion planning. *IEEE Transactions on Power Systems*, 12(1), 181 – 188.

58. Ghoseiri, K. and Nadjari, B. (2010). An ant colony optimization algorithm for the bi-objective shortest path problem. *Applied Soft Computing*, 10(4), 1237 – 1246.

59. Gilli, M., Kellezi, E. and Hysi, H. (2006). A data-driven optimization heuristic for downside risk minimization. *Journal of Risk*, 8(3), 1 – 18.

60. Glover, F. (1986). Future paths for Integer programming and links to Artificial Intelligence. *Computers and Operations Research*, 13, 533 – 549.

61. Glover, F. (1989). Tabu Search – Part I. *ORSA Journal on Computing*, 1, 190 – 206.

62. Glover, F. (1990). Tabu Search – Part II. *ORSA Journal on Computing*, 2, 4 – 32.

63. Glover, F. and Laguna, M. (1997). Tabu Search, *Kluwer Academic Publishers*, Dordrecht, The Netherlands.

64. Glover, F., Mulvey, J. M. and Hoyland, K. (1995). Solving dynamic stochastic control problems in finance using tabu search with variable scaling. In *Proceedings of the Metaheuristics International Conference, Kluwer Academic Publishers*, 429 – 448.

65. Goldberg, D. E. (1989). Genetic Algorithms in Search, Optimization, and Machine Learning. Reading, MA: *Addison Wesley*.

66. Goldreich, O. (2008). Computational Complexity: A Conceptual Approach. *Cambridge University Press*.

67. Hallow, W. V. (1991). Asset Allocation in a Downside-Risk Framework. *Financial Analyst Journal*, 47(5), 28 – 40.

68. Hamza, F. and Janssen, J. (1998). The Mean-Semivariances Approach to Realistic Portfolio Optimization subject to Transcation Costs. *Applied Stochastic Models and Data Analysis*, 14: 275 – 283.

69. Hassan, R., Cohanim, B., de Weck, O. and Venter, G. (2005). A comparison of Particle Swarm Optimization and the Genetic Algorithm. In *Proceedings of the 46th AIAA/ASME/ASCE/AHS/ASC Structures, Structural Dynamics and Materials Conference.*

70. Haupt, R. L. and Haupt, S. E. (2004). Practical Genetic Algorithms, 2nd ed. *John Wiley & Sons, Inc.*

71. Hedar, A. A. (2004). Studies on Metaheuristics for Continuous Global Optimization Problems. *PhD Thesis*, Kyoto University, Kyoto, Japan.

72. Henderson, D., Jacobson, S. H., and Johnson, A. W., (2003). The Theory and Practice of Simulated Annealing. In *Handbook of Metaheuristics*, F. Glover and G. A. Kochenberger (editors), International Series in Operations Research & Management Science, 57. *Kluwer Academic Publishers*, Dordrecht, The Netherlands.

73. Henderson, D., Vaughan, D. E., Jacobson, S. H., Wakefield, R. R. and Sewell, E. C. (2003). Solving the shortest route cut and fill problem using simulated annealing. *European Journal of Operational Research*, 145: 72 – 84.

74. Hertz, A. and de Werra, D. (1987). Using Tabu Search techniques for Graph Coloring. *Computing*, 39, 345 – 351.

75. Hertz, A. and de Werra, D. (1991). The Tabu Search metaheuristic: How we use it. *Annals of Mathematics and Artificial Intelligence*, 1, 111 – 121.

76. "Heuristic" A Dictionary of Computing (2004). Oxford Reference Online. http://www.oxfordreference.com/views/ENTRY.html?subview=Main&entry=t11.e2361.

77. Hogan, W. and Warren, J. (1974). Toward the Development of an Equilibrium Capital-Market Model Based on Semivariance, *Journal of Financial and Quantitative Analysis*, 9(1), 1 – 11.

78. Holland, J. H. (1975). Adaptation in Natural and Artificial Systems: An Introductory Analysis with application to Biology, Control, and Artificial Intelligence. *University of Michigan Press*, Ann Arbor, Michigan.

79. http://oxforddictionaries.com/view/entry/m_en_gb0840030#m_en_gb0840030 accessed on August 18th, 2010 at 11.25am.

80. Hu, N. (1992). Tabu Search method with random moves for globally optimal design. *International Journal for Numerical Methods in Engineering*, 35: 1055 – 1070.

81. Huang, X. (2008). Portfolio Selection with a new definition of risk. *European Journal of Operational Research*, 186, 351 – 357.

82. Jacob, C. and Khemka, N. (2004). Particle Swarm Optimization in Mathematica: An Exploration Kit for Evolutionary Optimization. *6th International Mathematica Symposium*, Banff, Canada (IMS 2004).

83. Jaszkiewicz, A. (2001). Multiple objective metaheuristic algorithms for combinatorial optimization. *Habilitation thesis*, Poznan University of Technology, Poznan.

84. Jobst, N.J., Horniman, M.D., Lucas, C.A., and Mitra, G. (2001). Computational Aspects of Alternative Portfolio Selection Models in the Presence of Discrete Assets Choice Constraints. *Quantitative Finance*, 1(5): 1 – 13.

85. Kannan, B. K. and Kramer, S. N. (1994). An augmented Lagrange multiplier based method for mixed integer discrete continuous optimization and its application to mechanical design. *ASME Transactions, Journal of Mechanical Design*, 116(2), 405 – 411.

86. Kendall, G. and Su, Y. (2005). A particle swarm optimization approach in the construction of optimal risky portfolios. In *Proceedings of the 23rd IASTED International Multi-Conference on Artificial Intelligence and Applications*, 140 – 145.

87. Kennedy, J. (1997). The particle swarm: Social adaptation of knowledge. In *Proceedings of the International Conference on Evolutionary Computation*, 303 – 308.

88. Kennedy, J. and Eberhart, R. C. (1995). Particle Swarm Optimization. In *Proceedings of the IEEE International Conference on Neural Networks*, 4: 1942 – 1948.

89. Kennedy, J., Eberhart, R. C with Shi, Y. (2001). Swarm Intelligence. *Morgan Kaufmann Publishers*, San Francisco, CA, USA.

90. Kirkpatrick, S., Gellatt, C. D. and Vecchi, M.P. (1983). Optimization by Simulated Annealing. *Science*, 220: 671 – 680.

91. Konno, H. and Yamazaki, H. (1991). Mean-Absolute Deviation Portfolio Optimization Model and Its Applications to Tokyo Stock Market. *Management Science*, 37: 519 – 531.

92. Koza J. R. (1995). Survey of genetic algorithms and genetic programming. In *Proceedings of 1995 WESCON Conference Record: Microelectronics, Communications Technology, Producing Quality Products, Mobile and Portable Power, Emerging Technologies*, pg 589 – 594. Piscataway, NJ: IEEE Service Center.

93. Lawler, E. L. (1976). Combinatorial Optimization: Networks and Matroids. *Holt, Rinehart, and Winston*, New York.

94. Lee, E. K. and Mitchell, J. E. (2000) Computational experience of an interior point SQP algorithm in a parallel branch-and-bound framework. In H Frenk *et al*, editor, *High Performance Optimization*, *Kluwer Academic Publishers*, Dordrecht, The Netherlands, pp 329 – 347.

95. Lee, S., von Allmen, P., Fink, W., Petropoulos, A. E. and Terrile, R. J. (2005). Comparison of Multi-Objective Genetic Algorithms in Optimizing Q-Law Low-Thrust Orbit Transfers. *GECCO'05 Late-breaking paper*, Washington, USA.

96. Leite, J. P. B. and Topping, B. H. V. (1999). Parallel simulated annealing for structural optimization. *Computers and Structures*, 73, 545 – 564.

97. Lin, C. K. Y. and Kwok, R. C. W. (2006). Multi-objective metaheuristics for a location-routing problem with multiple use of vehicles on real data and simulated data. *European Journal of Operational Research*, 175: 1833 – 1849.

98. Mansini, R. and Speranza, M. –G. (1999). Heuristic algorithms for the portfolio selection problem with minimum transaction lots. *European Journal of Operational Research*, 114(2): 219 – 233.

99. Mao, J. C. T. (1970). Models of Capital Budgeting, E-V vs E-S. *Journal of Financial and Quantitative Analysis*, 5(5), 657 – 676.

100. Maringer, D. (2005), Portfolio Management with Heuristic Optimization, *Springer-Verlag*, New York Inc.

101. Markowitz, H. M. (1952). Portfolio Selection. *Journal of Finance*, 7: 77 – 91.

102. Markowitz, H. M. (1956). The optimization of a quadratic function subject to linear constraints. *Naval Research Logistics Quarterly*, 3, 111 – 133.

103. Markowitz, H. M. (1959). Portfolio Selection: Efficient Diversification of Investments, *John Wiley*, New York, USA.

104. Markowitz, H. M. (1999). The early history of portfolio theory: 1600-1960. *Financial Analysts Journal*, 55(4): 5 – 16.

105. Markowitz, H., (1991). Portfolio Selection: Efficient Diversification of investments, 2nd ed, Cambridge, MA, *Basil Blackwell.*

106. Markowitz, H., Todd, P., Xu, G. and Yamane, Y. (1993). Computation of Mean-Semivariance Efficient Sets by the Critical Line Algorithm. *Annals of Operations Research*, 45(1): 307 – 317.

107. Metropolis, N., Rosenblutch, A., Rosenblutch, M., Teller, A. and Teller, E. (1953). Equation of state calculation by fast computing machines. *Journal of Chemical Physics*. 21: 1087 – 1092.

108. Michalewicz, Z. (1996). Genetic Algorithms + Data Structures = Evolution Programs (3rd ed). *Springer-Verlag*, New York.

109. Michalewicz, Z. and Fogel, D. B. (2004). How to Solve It: Modern Heuristics. *Springer-Verlag*, Berlin.

110. Misevičius, A., Blažaukas, T., Blonkis, J., and Smolinskas, J. (2004). An Overview of Some Heuristic Algorithms for Combinatorial Optimization Problems. *Informacinės Technologijos Ir Valdymas*, Nr.1(30).

111. Mitra, G., Kyriakis, T., Lucas, C. A. and Pirbhai, M. (2003). A Review of Portfolio planning: Models and Systems. An invited chapter In *Advances in Portfolio Construction and Implementation*, S. E. Satchell & A. E. Scowcroft (editors). *Butterworth & Heinemann*, Oxford.

112. Mohamed, C., Bassem, J. and Taicir, L. (2010). A genetic algorithms to solve the bicriteria shortest path problem. *Electronic Notes in Discrete Mathematics*, 36, 851 – 858.

113. Monticelli, A. J., Romero, R. and Asada, E. N. (2008). Fundamentals of Simulated Annealing. In *Modern Heuristic Optimization Techniques: Theory and Applications to Power Systems*, K. Y. Lee and Mohamed A. El-Sharkawi (editors), A *John Wiley & Sons*, Inc., Hoboken, New Jersey, USA.

114. Morgan, J. P. (1996). Risk Metrics, Technical Document, 4th *ed.*

115. Nawrocki, D. (1999). A brief history of Downside Risk Measures. *Journal of Investing*, 8(3), 9 – 25.

116. Onbasoglu, E. and Ozdamar, L. (2001). Parallel Simulated Annealing Algorithms in Global Optimization. *Journal of Global Optimization*, 19(1): 27 – 50.

117. OR Library: http://people.brunel.ac.uk/~mastjjb/jeb/orlib/files/.

118. Osman, I. H. and Laporte, G. (1996). Metaheuristics: A bibliography. *Annals of Operations Research*, 63, 513 – 623.

119. Perold, A. F. (1984). Large-Scale Portfolio Optimization. *Management Science*, 30(10): 1143 – 1160.

120. Quirk, J. P. and Saposnik, R. (1962). Admissability and Measurable Utility Functions. *Review of Economic Studies*.

121. Ram, D. J., Sreenivas, T. H. and Subramaniam, K. G. (1996). Parallel Simulated Annealing Algorithms. *Journal of Parallel and Distributed Computing*, 37(2), 207 – 212.

122. Ramachandran, K. M. and Tsokos, C. P. (2009). Mathematical Statistics with Applications. *Elsevier Academic Press*, San Diego, California, USA.

123. Reeves, C. R. and Beasley, J.E (1995). Introduction. In *Modern Heuristic Techniques for Combinatorial Problems*, C. R. Reeves (editor). *McGraw-Hill International* (UK) Limited.

124. Reeves, C. R. and Rowe, J. E. (2002). Genetic Algorithms–Principles and Perspectives: A Guide to GA Theory, Operations Research/Computer Science Interfaces Series, Vol. 20, *Springer Berlin Heidelberg* New York.

125. Reynolds, C. (1987). Flocks, herds, and schools: A distributed behavioral model. *Computer Graphics*, 21(4), 25 – 34.

126. Rolland, E. (1997). A tabu search method for constrained real number search: applications to portfolio selection. *Technical report, Department of Accounting and Management Information Systems*, Ohio State University, Columbus, USA.

127. Roy, A. D. (1952). Safety first and the holding of assets. *Econometrics*, 20, 431 – 449.

128. Sandgren, E. (1988). Nonlinear integer and discrete programming in mechanical design. In *Proceedings of the ASME Design Technology Conference*, Kissimine, Fl; 95 – 105.

129. Schaerf, A. (2002). Local Search Techniques for Constrained Portfolio Selection Problems. *Computational Economics*, 20: 177 – 190.

130. Shi, Y. and Eberhart, R. C. (1998). A Modified Particle Swarm Optimizer. In *IEEE International Conference of Evolutionary Computation.*

131. Shi, Y. and Eberhart, R. C. (1999). Empirical Study of Particle Swarm Optimization. In *Proceedings of the Congress on Evolutionary Computation*, 1945 – 1949.

132. Siarry, P. and Berthiau, G. (1997). Fitting of Tabu Search to optimize functions of continuous variables. *International Journal for Numerical Methods in Engineering*, 40: 2449 – 2457.

133. Silver, E. A. (2004). An overview of heuristic solution methods. *Journal of the Operational Research Society*, 55: 936 – 956.

134. Speranza, M. G. (1996). A heuristic algorithm for a portfolio optimization model applied to the Milan stock market. *Computers & Operations Research*, 23(5): 433 – 441.

135. Streichert, F., Ulmer, H. and Zell, A. (2003). Evolutionary Algorithms and the Cardinality Constrained Portfolio Optimization Problem. In D. Ahr, R. Fahrion, M. Oswald, and G. Reinelt, editors, Operations Research Proceedings 2003, *Selected Papers of the International Conference on Operations Research*, Heidelberg, Springer.

136. Suman, B., Hoda, N. and Jha, S. (2010). Orthogonal simulated annealing for multiobjective optimization. *Computers and Chemical Engineering*, 34(10), 1618 – 1631.

137. Suppapitnarm, A., Seffen, K. A., Parks, G.T. and Clarkson, P. J. (2000). A Simulated annealing: An alternative approach to true multiobjective optimization. *Engineering Optimization*, 33, 59 – 85.

138. Taillard, E. D. (1990). Some efficient heuristic methods for the flow shop sequencing problem. *European Journal of Operational Research*, 47(1), 65 – 74.

139. Taillard, E. D. (1991). Robust taboo search for the quadratic assignment problem. *Parallel Computing*, 17, 443 – 455.

140. Taillard, E. D. (1993). Parallel iterative search methods for vehicle routing problems. *Networks*, 23, 661 – 673.

141. Taillard, E. D. (1994). Parallel taboo search techniques for the job shop scheduling problem. *ORSA Journal on Computing*, 6(2), 108 – 117.

142. Talbi, E-G. (2009). Metaheuristics: From Design to Implementation. *John Wiley & Sons Inc.*, Hoboken, New Jersey.

143. Test Problems for Constrained Global Optimization available at: http://www-optima.amp.i.kyoto-u.ac.jp/member/student/hedar/Hedar_files/TestGO_files/Page422.htm.

144. Trosset, M. W. (2009). An Introduction to Statistical Inference and its Applications with R. *Chapman & Hall/CRC*, Boca Raton, FL.

145. Uryasev, S. and Rockafellar, R. T. (1999). Optimization of conditional Value-at-Risk, Research Report, ISE Department, University of Florida, USA.

146. van den Bergh, F. (2001). An Analysis of Particle Swarm Optimizers. *Ph D. Thesis*, University of Pretoria, Pretoria, South Africa.

147. Wang, X. H. and Li, J. J. (2004). Hybrid Particle Swarm Optimization with Simulated Annealing. In *Proceedings of the Third International Conference on Machine Learning and Cybernetics*, pp. 2402 – 2405.

148. Weise, T. (2009). Global Optimization Algorithms – Theory and Application. Available at *www.it-weise.de/projects/book.pdf* and accessed on 19th November, 2010; 6:10am.

149. Wright, MB (2003). An Overview of neighbourhood search metaheuristics. Working paper, the Department of Management Science, Lancaster University, UK.

150. Xiang, Z., Chu, C. and Chen, H. (2006). A fast heuristic for solving a large-scale static dial-a-ride problem under complex constraints. *European Journal of Operational Research*, 174: 1117 – 1139.

151. Xiang, Z., Chu, C. and Chen, H. (2008). The study of a dynamic dial-a-ride problem under time-dependent and stochastic environments. *European Journal of Operational Research*, 185: 534 – 551.

152. Yahaya, A. and Wright, M. (2010). SWAN – A hybridized approach for solving Portfolio Selection Problems. In *Proceedings of the 2nd Student Conference on Operational Research (SCOR2010)*, pp 69 – 73.

153. Yang, X. –S. (2008). Introduction to Mathematical Optimization – From Linear Programming to Metaheuristics. *International Cambridge International Science Publishing*, Cambridge, UK.

154. Yang, X. –S. (2010). Engineering Optimization: An Introduction with Metaheuristic Applications. *John Wiley and Sons*.

155. Yang, X. –S. and Deb, S. (2010). Engineering Optimization by Cuckoo Search. *International Journal of Mathematical Modelling and Numerical Optimization*, 1(4), 330 – 343.

156. Zitzler, E. (1999). Evolutionary algorithms for multiobjective optimization: Methods and applications, *PhD thesis*, Swiss Federal Institute of Technology Zurich.

157. Zitzler, E., Deb, K. and Thiele, L. (2000). Comparison of multiobjective evolutionary algorithms: Empirical results. *Evolutionary Computation*, 8(2), 173 – 195.

158. Zitzler, E., Laumanns, M. and Thiele, L. (2001). SPEA2: improving the strength Pareto evolutionary algorithm. *Swiss Federal Institute of Technology*, Tech Rep: 103.

APPENDIX 1

Endogenous

In this section, we want to illustrate (with a very small numerical example) the endogenic nature of a semicovariance matrix (which is one of the vital inputs in the E-S portfolio optimization problem) computed using equation *2.5.1(d)* as suggested by Markowitz [103].

Consider a set of two fictitious assets (*Asset1* & *Asset2*) with annual returns expressed in % (D6:E15) over a ten-year period from 1997 through 2006. Columns F(F6:F15), G(G6:G15) and H(H6:H15) in Table I below respectively represent portfolio return values for three different portfolios in which the first invested 80% in *Asset1* and 20% in *Asset2*, the second invested 40% in *Asset1* and 60% in *Asset2*; while the last invested 10% in *Asset1* and the remaining 90% in *Asset2*.

Proving the Endogeneity of Semicovariance Matrix computed based on Markowitz' [2] suggestion as in equation 2.5.1(d)

Benchmark, B = 0.01

s/no	YEAR	Asset1	Asset2	Returns for diff Portfolio Configurations 80% - 20%	40% - 60%	10% - 90%	80% - 20% Portfolio CondRet_Asset1	CondRet_Asset2	Product	40% - 60% Portfolio CondRet_Asset1	CondRet_Asset2	Product	10% - 90% Portfolio CondRet_Asset1	CondRet_Asset2	Product
1	1997	0.310	-0.212	0.206	-0.003	-0.160	0.000	0.000	0.000	0.300	-0.222	-0.067	0.300	-0.222	-0.067
2	1998	0.267	-0.093	0.195	0.051	-0.057	0.000	0.000	0.000	0.000	0.000	0.000	0.257	-0.103	-0.026
3	1999	0.195	0.368	0.230	0.299	0.351	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
4	2000	-0.101	-0.272	-0.135	-0.204	-0.255	-0.111	-0.282	0.031	-0.111	-0.282	0.031	-0.111	-0.282	0.031
5	2001	-0.130	-0.235	-0.151	-0.193	-0.225	-0.140	-0.245	0.034	-0.140	-0.245	0.034	-0.140	-0.245	0.034
6	2002	-0.234	-0.186	-0.224	-0.205	-0.191	-0.244	-0.196	0.048	-0.244	-0.196	0.048	-0.244	-0.196	0.048
7	2003	0.264	0.245	0.260	0.253	0.247	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
8	2004	0.090	0.076	0.087	0.082	0.077	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
9	2005	0.030	0.402	0.104	0.253	0.365	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
10	2006	0.136	0.069	0.123	0.096	0.076	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000

Endogenous SemiCov Matrix based on equation 2.5.1(d)	0.009	0.011 0.018	0.018	0.005 0.023	0.025	0.002 0.024
Portfolio SemiVariance based on equation 2.5.1(h)	0.010		0.013		0.020	

Figure 41: Proving the endogeneity of semicovariance matrix

Now suppose the benchmark return (cell L3), B = 1%. Table I shows among other things, the conditional returns of *Asset1* (CondRet_Asset1) [cells K6:K15] computed

using equation *2.5.1(d)* (Chapter Two). When any of the return values of the 80% – 20% portfolio [cells F6:F15] outperformed the benchmark (B = 0.01), the corresponding conditional return value will be 0. But when the portfolio's return value is outperformed by benchmark, the corresponding conditional return value will definitely take a negative value [Recall $(R_{it} - B)$ in equation 2.5.1(d)]. To clarify what has been said; the conditional return of *Asset1* takes a value of 0.0 (cell K7) in 1998, because the 80% – 20% portfolio yielded a return value of 0.195 (cell F7), thereby outperforming the benchmark; similarly, it takes a value of –0.14 (cell K10) in 2001 [Recall $(R_{it} - B)$], because the 80% – 20% portfolio yielded a return value of –0.151 (cell F10), thereby underperforming the benchmark. Similar approach was executed to obtain conditional returns for *Asset2* (cells L6:L15); and the next column (cells M6:M15) was just the *product* of the two preceding columns (K6:K15 and L6:L15).

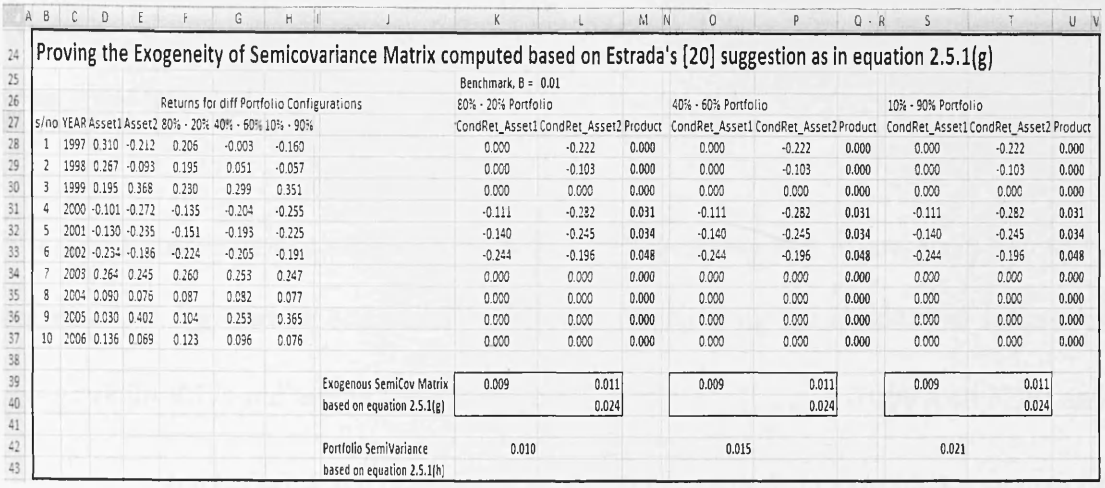
We now explain how we compute the elements of the Endogenous Semicovariance matrices (K17:T18) from equations *2.5.1(c)* and *2.5.1(d)*. Sticking to the 80% – 20% portfolio example; now by squaring the conditional returns of *Asset1* and taking their average [refer to equation *2.5.1(c)*] we obtain $S^2_{Asset1\,(0.01)} = 0.009$ (cell K17). By executing similar operations with conditional returns of *Asset2* we obtain $S^2_{Asset2\,(0.01)} = 0.018$ (cell L18), while $S_{Asset1\,Asset2\,(0.01)} = 0.011$ (cell L17) results directly from equation *2.5.1(d)*. Thus, it follows from equation 2.5.1(h) that the semivariance of the 80% – 20% portfolio (cell K20) is: $\left\{(0.8)^2(0.009) + (0.2)^2(0.018) + 2(0.8)(0.2)(0.011)\right\} = 0.010$. The corresponding values for 40% – 60% portfolio (cell O20) and 10% – 90% portfolio (cell S20) are respectively 0.013 and 0.020.

Now, it is clear just from the two-asset example considered, the semicovariance matrices, although resulting from the same expressions [equations *2.5.1(c)* and *2.5.1(d)*] and same set of asset returns (cells D6:E15) are remarkably different and this is because their elements depend on the asset weights, hence endogenous.

Exogenous

In this section, we want to illustrate why the semicovariance matrices obtained using Estrada's [49] heuristic as in equation *2.5.1(g)* are exogenic. Table II below reproduces the returns over the 1997 – 2006 of Asset1, Asset2, 80% – 20% portfolio, 40% – 60% portfolio and 10% – 90% portfolio all taken from Table I above. As already shown above, the elements of the semicovariance matrices that result from equations *2.5.1(c)* and *2.5.1(d)* for the 80% – 20% portfolio are different from those of the 40% – 60% portfolio as well as those of the 10% – 90% portfolio, thereby confirming the endogeneity of the Markowitz' definition of semicovariance.

Recall that, with Markowitz definition of semicovariance, the knowledge of whether *portfolio's return* (and ***not*** *assets' return*) underperformed the benchmark B is required (thereby generating the endogeneity problem discussed above). However, with Estrada's definition too, knowledge of whether *assets'* (*and* ***not*** *portfolio's*) *return* underperformed the benchmark B is needed; and as will be shown the resultant semicovariance matrices are invariant of the portfolio configuration considered, and are thus symmetric as well as exogenic.



Proving the Exogeneity of Semicovariance Matrix computed based on Estrada's [20] suggestion as in equation 2.5.1(g)

Benchmark, B = 0.01

s/no	YEAR	Asset1	Asset2	80% - 20%	40% - 60%	10% - 90%
		Returns for diff Portfolio Configurations				
1	1997	0.310	-0.212	0.206	-0.003	-0.160
2	1998	0.267	-0.093	0.195	0.051	-0.057
3	1999	0.195	0.368	0.230	0.299	0.351
4	2000	-0.101	-0.272	-0.135	-0.204	-0.255
5	2001	-0.130	-0.235	-0.151	-0.193	-0.225
6	2002	-0.234	-0.186	-0.224	-0.205	-0.191
7	2003	0.264	0.245	0.260	0.253	0.247
8	2004	0.090	0.076	0.087	0.082	0.077
9	2005	0.030	0.402	0.104	0.253	0.365
10	2006	0.136	0.069	0.123	0.096	0.076

	80% - 20% Portfolio			40% - 60% Portfolio			10% - 90% Portfolio		
	CondRet_Asset1	CondRet_Asset2	Product	CondRet_Asset1	CondRet_Asset2	Product	CondRet_Asset1	CondRet_Asset2	Product
	0.000	-0.222	0.000	0.000	-0.222	0.000	0.000	-0.222	0.000
	0.000	-0.103	0.000	0.000	-0.103	0.000	0.000	-0.103	0.000
	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
	-0.111	-0.282	0.031	-0.111	-0.282	0.031	-0.111	-0.282	0.031
	-0.140	-0.245	0.034	-0.140	-0.245	0.034	-0.140	-0.245	0.034
	-0.244	-0.196	0.048	-0.244	-0.196	0.048	-0.244	-0.196	0.048
	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000

Exogenous SemiCov Matrix	0.009	0.011	0.009	0.011	0.009	0.011
based on equation 2.5.1(g)		0.024		0.024		0.024
Portfolio SemiVariance based on equation 2.5.1(h)	0.010		0.015		0.021	

Figure 42: Proving the exogeneity of semicovariance matrix

By considering again a benchmark return, B of 1% (cell L25), we can compute the four elements of the 80% – 20% portfolio semicovariance matrix as follows: We now redefine the conditional returns [CondRet_Asset1 (K28:K37) & CondRet_Asset2 (L28:L37)] to take a value of 0.0 when the corresponding asset return is larger than (thus outperforming) the benchmark B; and to take the value of the difference $(R_{it} - B)$ when the corresponding asset return is smaller than (thus underperforming) the benchmark B.

To clarify further, the conditional return of *Asset1* in 1997 takes the value 0.0 (cell K28), due to the fact that, *Asset1* delivered a return value of 0.31 (cell D28) thereby outperforming the benchmark; however, in 2002 it took a negative value $(R_{it} - B)$ of –0.244 (cell K33), because the same *Asset1* now yielded a negative return of –0.234 (cell D33) thereby underperforming the benchmark B of 1%. It is important to note that, because these conditional returns (for both *Asset1* and *Asset2*) depend on their corresponding original asset (***and not portfolio***) returns underperforming the benchmark, they are very relevant not only to the 80% – 20% portfolio, but to other

portfolios of any kind of configuration (whether it is 10% – 90%, 50% – 50% and/or even 1% – 99%).

The four semicovariance terms for the 80% – 20% portfolio that follow from Estrada's equation *2.5.1(g)* can be computed as follows: By squaring the conditional returns of *Asset1* (K28:K37) and taking their average we obtain $S^2_{Asset1\,(0.01)} = 0.009$ (cell K39) and $S^2_{Asset2\,(0.01)} = 0.024$ (cell L40) is obtained in similar fashion from conditional returns of *Asset2*; while $S_{Asset1\,Asset2\,(0.01)} = 0.011$ (cell L39) is obtained by taking the average of the elements in the *product* column (M28:M37). As described in illustrating endogeneity of the semicovariance matrix, we can, in similar way, compute [from equation *2.5.1(h)*] the semivariance of the 80% – 20% portfolio (cell K42 by: $\left\{(0.8)^2(0.009)+(0.2)^2(0.024)+2(0.8)(0.2)(0.011)\right\} = 0.010$. The corresponding values for 40% – 60% portfolio (cell O42) and 10% – 90% portfolio (cell S42) are respectively 0.015 and 0.021.

It is important to note that, the semicovariance matrices for the 80% – 20% portfolio, 40% – 60% portfolio and the 10% – 90% portfolio *have te same number of elements with all corresponding terms being equal* and the only difference is in the different weights allotted to the assets in different portfolio configurations. It is also equally important to note that, the values for the portfolio semivariance using Estrada's approximation are either equal (K42 = K20) or very close (O42 ≅ O20 & S42 ≅ S20) to the actual values they tend to estimate.

APPENDIX 2

Justification for algorithms parameter choice

Although, algorithmic parameter settings are problem-dependent; we decided to run our algorithms several number of times in order to have an idea about the best parameter choice for our implemented algorithms. The figures and tables under each algorithm show the results obtained after such experimental runs were conducted. In each case, we reported the time taken (in seconds) for the algorithm to generate an entire *UEF* (consisting of 200 portfolios). We further computed the *mEd* to measure the distance between the *UEF* generated by a particular setting and the optimal *UEF* generated by a non-linear optimization solver – CPLEX 11.2. In all cases, the smaller the numerical values of these measures, the better the algorithmic settings.

SA parameter choice decisions

Going by the fact that SA has several parameters needing proper and careful tuning, we decided to run the algorithm several times with different parameter settings in order to come up with a set of parameter combinations that seem to be able to estimate the (approximate) PSP Pareto front better, and in a computationally reasonable time frame.

In order to come up with parameter combinations that will produce good results for our SA implementation, we decided to run the simulation several times using different set of parameter combinations. The set of parameters collectively known as cooling schedule comprises of the Initial temperature (T_0), the cooling rate (α), the length of Markov chain (N) and the final temperature (T_f). We tested several cooling schedules, but due to time and space constraints, we are only able to report the performance of

five of them. In all the five cooling schedules (CS1 – CS5) reported, the initial and final temperatures were pegged at **1.0** and **0.001** respectively; while the cooling rate and length of the Markov chain were made to vary such that in all cases, the total number of iterations will be approximately **6600** (a little bit more than twice the number of iterations in our multi-agents techniques).

The following table gives a summary of the settings in each cooling schedule.

Cooling Schedules	Cooling rate, α	Markov Chain, N	Total iterations
CS1	0.99063	9	6606
CS2	0.9	100	6600
CS3	0.8	213	6603
CS4	0.5	660	6600
CS5	0.25	1320	6600

Table 12: Summary of different settings on cooling schedules

The figure and table below show the results obtained by running SA using the above five mentioned cooling schedules.

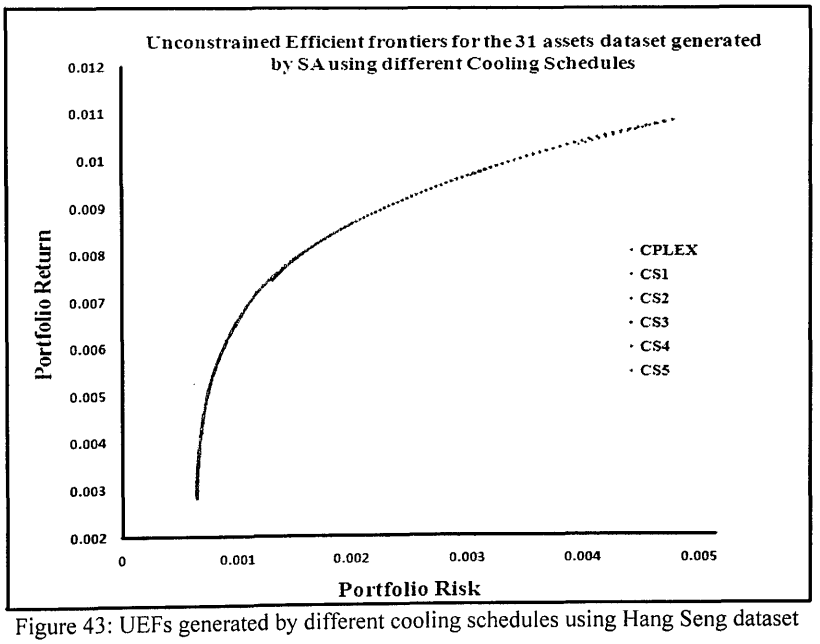


Figure 43: UEFs generated by different cooling schedules using Hang Seng dataset

Cooling Schedules	Time (s)	mEd
CS1	298.9	4.96E-06
CS2	305.8	5.99E-06
CS3	300.5	7.89E-06
CS4	323.0	9.74E-06
CS5	326.3	6.91E-06

Table 13: Summary of objective values and time taken by different cooling schedules

From the figure above, mere eyeballing is not sufficient to reveal any difference whatsoever between the different *UEF*s generated by CPLEX and SA (using the 5 cooling schedules – CS1 through CS5); however, the neighbouring table shows some interesting results. Although, there is not much significant difference in the performance of different cooling schedules; it can easily be noticed that: CS1 marginally outperformed the other cooling schedules (on the average) in terms of the *mEd* and the total time taken to generate a frontier of efficient portfolios. It is also interesting to note that, our results agree with and proved the importance of the suggestion made by Eglese [43] that: it is important for the (SA) algorithm to spend less time at extreme (higher and lower) values of the control (temperature) parameter; and this is exactly what our CS1 does, as the numerical value of its length of Markov chain is just 9 unlike the other CS whose length is more than 100 in each case, thereby wasting much time at both extremes of the temperature parameter.

Now going by the superiority of the average performance of CS1 over other cooling schedules (CS2 through CS5); we decide to implement all SA's experimental runs with this cooling schedule.

Parallel SA parameter choice decisions

This search method operates in similar manner as the SA, the only difference lies in the number of solutions dispersed over entire search space, hence parallel SA. Therefore, all the decisions (*generic* and *problem-specific*) reached in relation to the SA are as well adopted in this algorithm. However, we decided to fix the number of (parallel) solutions to the magnitude of the size (dimension) of our problem. So for our 31 asset dataset, we generated 31 parallel solutions and each considers *neighbours* twice the dimension of the problem. With this method, we hope to obtain solutions that are *at least* as good as those obtained by SA.

TS parameter choice decisions

(i) *Tabu Tenure*: we tried some different set of values for the tabu tenure based on the empirical evidence obtained by running our algorithm quite a number of times. The reason behind choosing our tabu tenure value can be seen in the following chart and table

S/No	Tabu Tenure	Time (s)	mEd
1	3	194.3	4.51E-04
2	5	207.1	5.79E-05
3	7	235.2	1.29E-06
4	11	271.1	1.02E-06
5	17	324.3	1.12E-07
6	21	375.0	1.00E-08

Table 14: Summary of objective values and time taken for different sizes of tabu tenure

(ii) *Tabu Region*: After several independent runs we arrived at a reasonable Euclidean distance threshold value (between two neighbouring solutions) of $\mathbf{10^{-5}}$.

(iii) *Total Number of Neighbours considered*: This is set as in section 4.2.2.1 above.

PSO parameter choice decisions

(i) ***The Acceleration Coefficients***: The following figure and table show the results obtained after running the PSO algorithm a number of times to reveal the effect of changing different values of acceleration coefficients (C_1 & C_2).

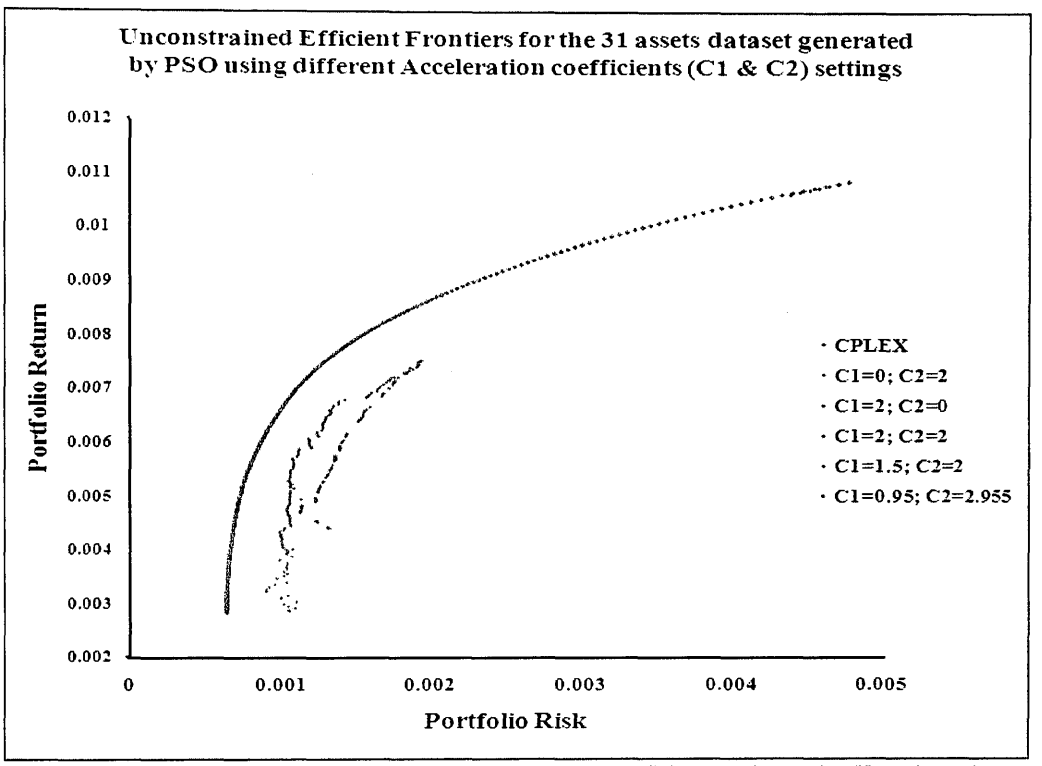


Figure 44: UEFs generated by PSO using different acceleration coefficients settings using Hang Seng dataset

Acceleration coefficients	Time (s)	mEd
C1=0; C2=2	151.4	1.29E-06
C1=2; C2=0	190.0	1.77E-03
C1=2; C2=2	209.8	1.29E-07
C1=1.5; C2=2	196.3	1.45E-07
C1=0.95; C2=2.955	189.5	1.17E-07

Table 15: Summary of objective values and time taken for different settings of acceleration coefficients

From Figure 45 above, it is hard to notice any difference in the *UEF* generated by 4 different settings other that of C_1=2 and C_2=0 (green points) which plainly deviates from the optimal and the remaining *UEFs*. However, from the table, it can be seen that, the worst performance of the algorithm in terms of (*mEd*) occurred when $C_1 = 2$, and $C_2 = 0$, and this setting literally means an implementation of *cognition only* PSO, because all particles are attracted to only their personal best solutions without the potential of exploring the entire search space; and each particle has no idea what or where the global best solution is positioned on the search space and neither does any particle's movements get influenced in anyway by the global best solution; hence, the reason for the poor performance. On the other hand, the implementation of the *social only* PSO (where $C_1 = 0$ and $C_2 = 2$), in which all particles cooperate with one another by using information from the global best solution to continue with their search trajectories (without following any personal experience) was found to be superior (in performance) than the *cognitive only* implementation, and this result is in agreement with what was obtained by Kennedy [87].

Kennedy and Eberhart [88] in their original PSO implementation suggested for acceleration coefficients' setting of $C_1 = C_2 = 2$. Based on our PSP, this setting yielded a better result than either the *cognition* or *social* only implementations, with an *mEd* = 1.29E-07 taking approximately 210secs to generate the entire *UEF* of 200 portfolios; and this (we believe) is not unconnected with the fact that: although, the *social* factors seemingly, by empirical evidence, play more important role (in PSO's performance) than their *cognitive* counterparts; the combination of both (*cognitive* and *social*) components play even more important role in PSO's successful implementation.

We further explored the potential of our algorithm, by trying different settings to the one suggested in literature, in which we notice an improvement in the algorithm's performance when the *cognitive* (C_1) factor takes a value slightly less than one and the *social* factor (C_2) takes a value greater than two. After different settings were tried and several experimental runs conducted, we finally settled on: **C_1=0.95 and C_2=2.955**. As can be seen from the table above, this setting provided a better result (in terms of *mEd*) in comparison to all the results reported therein and faster (in terms of time taken) in comparison to the results provided by the three settings just above it in the table. It provided the smallest value of *mEd* = 1.17E-07 and was found to take on average less than one second for each of the generated points on the *UEF* (i.e 189.5 secs for 200 points on the *UEF*).

(ii) ***The Inertia Weight***:

The Figure and table below shows the effect of changing inertia weight, w, in different experimental runs to generate *UEF*.

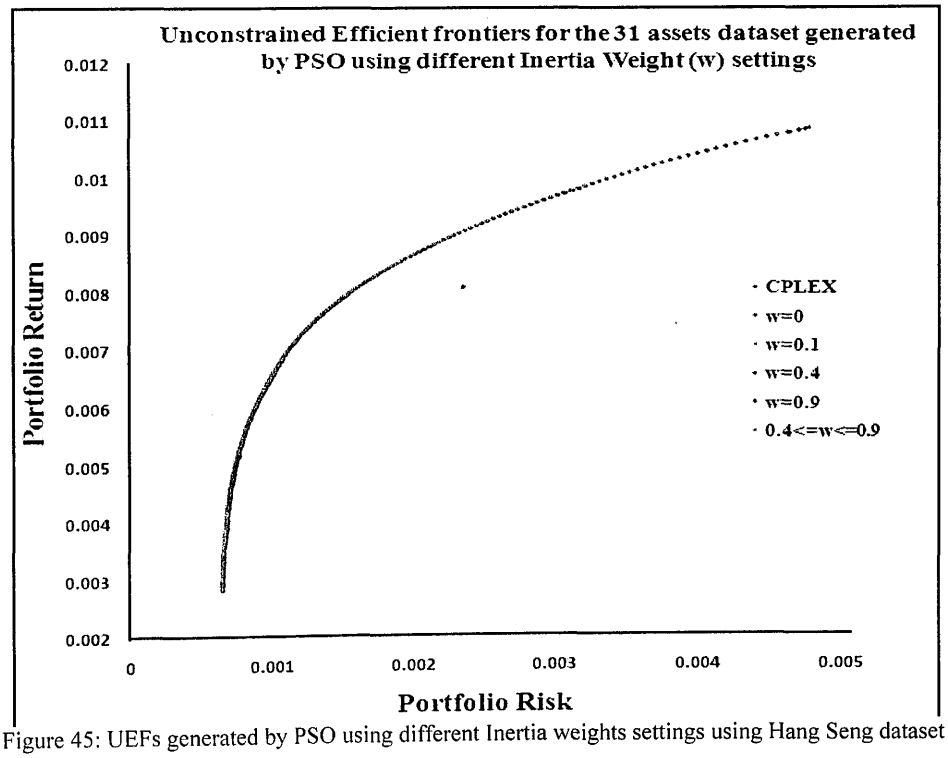


Figure 45: UEFs generated by PSO using different Inertia weights settings using Hang Seng dataset

Inertia Weight, w	Time (s)	mEd
w = 0	49.0	2.79E-03
w = 0.1	155.5	1.28E-05
w = 0.4	171.7	3.39E-06
w = 0.9	197.3	9.95E-07
0.4<=w<= 0.9	190.1	3.19E-07

Table 16: Summary of objective values and time taken for different settings of Inertia Weights

From Figure 45 above, the only *UEF* that deviates from the optimal one (generated by CPLEX 11.2) was the one generated by setting $w = 0$ (see red points). This setting literally means, particles' decisions on their next position in the search space is not in any way governed by the previous velocity in the previous time step. In agreement with the results of Shi and Eberhart [130], it is easily noticeable from the above table that, as the inertia weight, w value approaches unity; there is an apparent improvement of the algorithm's performance, though with an accompanying time-cost consequences. From the table, it can easily be seen that, there is a steady improvement in the *mEd* values (2.79E-03 to 1.28E-05 to 3.39E-06 to 9.94E-07) for the respective settings when w takes values: $w = 0$, $w = 0.1$, $w = 0.4$ & $w = 0.9$.

Going by the suggestion of Kendall and Su [86] and in order to allow our algorithm to properly explore a very large area of the search space at the beginning of the simulation runs and to further refine the search at later stage, we decided to adopt the dynamic approach in which the inertia weight, w, initially takes the maximum value of 0.9, and as the search progresses it takes different values within the real interval [0.4, 0.9] up to the point where it takes the minimum value of 0.4. This setting was found,

among other different settings tried, to be good in estimating the UEF as well as reasonable in the amount of time taken to generate such a frontier.

Going by the fact that, the dynamic setting (in which the inertia factor, w, takes values within a real interval [**0.4**, **0.9**] inclusive) provided a better result than all other results reported in the above table; we decided to adopt such setting in all our PSO implementations (both constrained and unconstrained).

(iii) ***Particles' size***: Particles' size undoubtedly is a parameter to reckon with in any PSO implementation. In our PSO implementation, we tested several particles' sizes involving 20, 31 (equivalent to the problem's dimension), 50, 62 (twice the problem's dimension), and 93 (thrice the problem's dimension) for the 31 assets dataset.

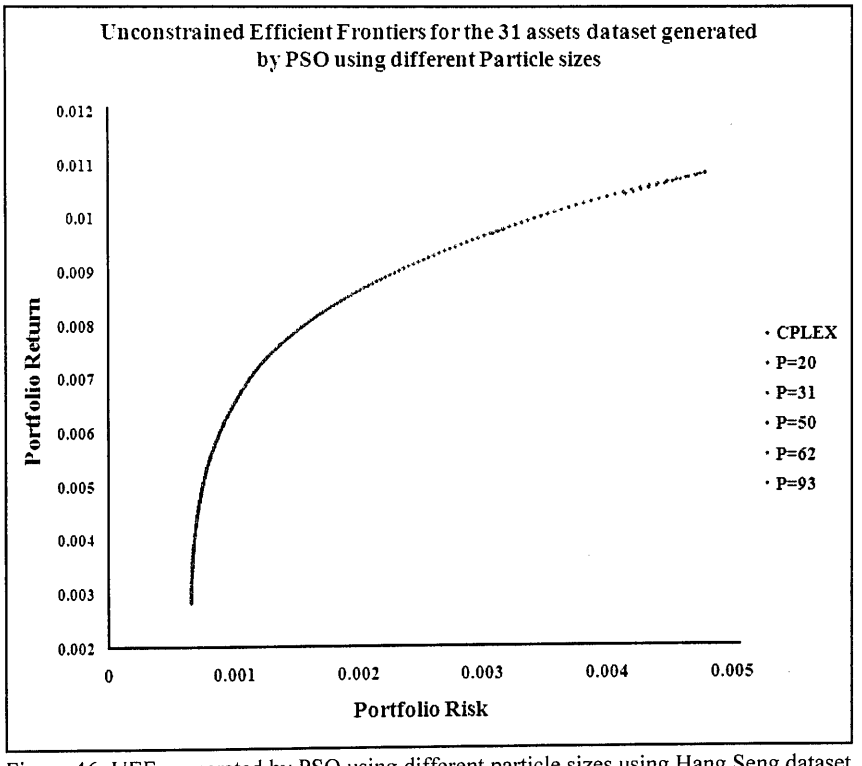


Figure 46: UEFs generated by PSO using different particle sizes using Hang Seng dataset

Particles Size	Time (s)	mEd
P = 20	76.9	3.34E-06
P = 31	117.7	1.64E-06
P = 50	188.1	6.66E-07
P = 62	236.5	8.16E-08
P = 93	351.9	4.71E-08

Table 17: Summary of objective values and time taken for different particles' sizes

The UEFs generated in the figure above using the five different particles' sizes settings cannot be easily distinguished, as they all seem to coincide with the one generated by the nonlinear optimization solver (CPLEX 11.2). However, the neighbouring table reveals the difference in performance (in terms of mEd) and time taken.

From the above table, it can easily be noticed that, performance improves with increase in the number of particles; however, this achievement has an accompanying costly consequences. This is because, the number of particles seems to be positively correlated strongly with the total time taken. Based on the result reported, we settled on a size of 50 particles in both constrained and unconstrained PSO implementations. This setting was found to be sufficient in obtaining near-optimal solutions, yet at a very reasonable time frame.

SWAN parameter choice decisions

As this method comes into being as a result of hybridizing PSO and SA, it is not strange to see that it combined parameters of both algorithms. We decided to run the algorithm with different number of settings in order to determine which one would be more appropriate in generating a UEF within a reasonable time frame. Although, we

tried different settings, we decided to report only 5 among them out of which we consider the best in our implementation. Each setting comprises of five different parameter choices, namely: the acceleration coefficients (C_1 & C_2), inertia weight (w), PSO total iterations, SA's cooling rate, α and length of Markov chain, N_k.

The following table shows the different parameter choices under the 5 SWAN parameter settings

SWAN Parameter settings					
Settings	Acceleration coefficients	Inertia Weight	PSO iterations	Cooling rate, α	Markov Chain, N
set1	C1=0.95; C2=2.955	0.4<=w<= 0.9	2266	0.99063	1
set2	C1=0; C2=2	w = 0	2274	0.9	11
set3	C1=2; C2=0	w = 0.1	2287	0.8	23
set4	C1=2; C2=2	w = 0.4	2290	0.5	71
set5	C1=1.5; C2=2	w = 0.9	2290	0.25	142

Table 18: Summary of different SWAN parameter settings

The SWAN parameter settings depicted in the figure above were meant to make sure that in the entire experimental run, a total of **3000** (for both PSO and SA parts) iterations are conducted as in other multi-agent techniques such as the PSO, GA and parallel SA.

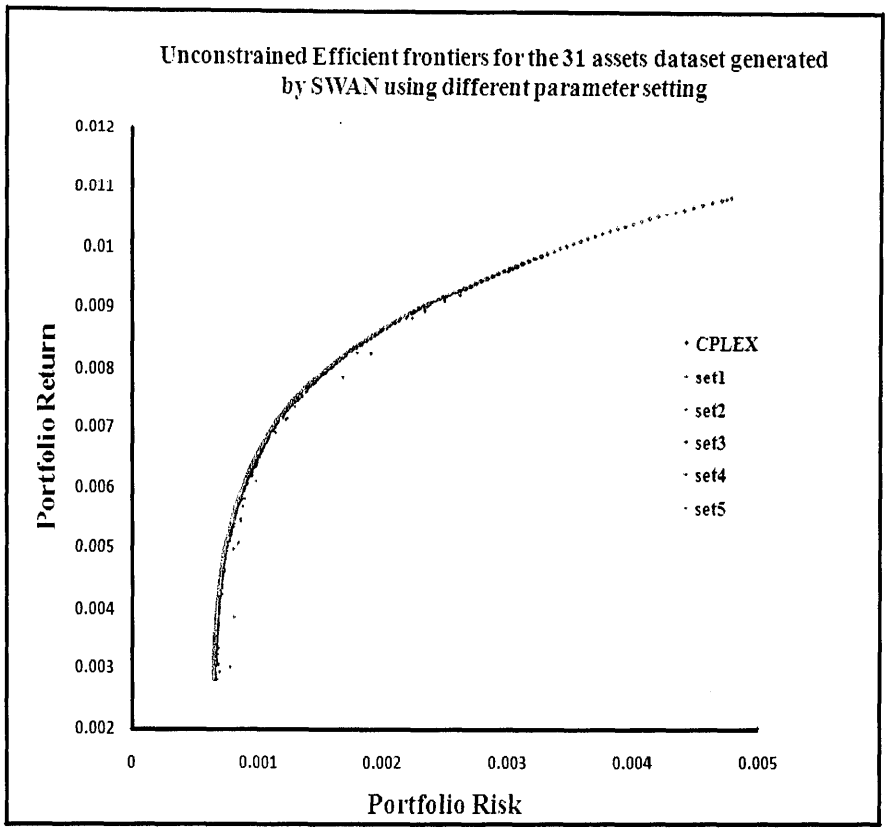


Figure 47: UEFs generated by SWAN using different parameter settings using Hang Seng dataset

Settings	Time (s)	mEd
set1	329.7	1.00E-07
set2	195.4	2.45E-05
set3	213.5	2.47E-05
set4	313.8	1.81E-06
set5	230.5	7.77E-07

Table 19: Summary of objective values and time taken for different SWAN parameter settings

Like other comparative *UEF* plots in this section; If not for some few number of points that deviated from the main frontier, it is difficult to distinguish the *UEF* plots generated by different settings. However, the neighbouring table reveals that the best performing setting is **set1** with *mEd* of **1.00E-07**.

GA parameter choice decisions

(i) *Population size*: This refers to the total number of individuals that are initially begin with and continued to be maintained throughout the search history. These are synonymous to *particles* and *parallel solutions* in PSO and parallel SA implementations respectively. During the initial implementation of this algorithm we tried a population size of 100 (more than 3 times the dimension of our smaller dataset), but as we kept on improving it, we found that as few as 31 individuals often provide very competitive solutions.

(ii) *Generations*: This is synonymous to the total number of iterations in other search methods. So to keep in tune with other algorithms, we set the total number of generations to complete a cycle at 2700, but after several simulation runs we found out that 1000 generations is enough to provide a very good solution.

(iii) *Genetic Operators*: A typical GA uses three to four basic operators: *selection, crossover, mutation* and *elitism* to direct the population of individuals towards convergence to a global optimum. These operators are discussed below:

(a) *Selection*: Although, there are several ways in which this operation can be executed, for this research we found *roulette-wheel selection* approach (which is proportional to the fitness of an individual) more convenient to our type of problem.

(b) *Crossover*: The following figure and table justifies the choice of the type of crossover and the corresponding probability that seem to give an

approximate estimate of the *UEF*.

(c) *Mutation*: The following figure and table justifies the choice of the type of the probability of mutation that seem to give an approximate estimate of the *UEF*.

(d) *Elitism*: We decide to always carry fittest individuals amounting to 10% of the entire population size to the next generation as part of our elitism operation.

(iv) *Population replacement*: As in Chang *et al* [20], we employ a steady-state population replacement approach, in which pair of newly *born* children replaces a pair of less-fit members of the old population and the process continues until the desired population size is attained.