ENHANCING THE BEES ALGORITHM FOR GLOBAL OPTIMISATION USING SEARCH SPACE MANIPULATION

by

TURKI BIN BAKIR

A thesis submitted to the University of Birmingham for the degree of

DOCTOR OF PHILOSOPHY

Department of Mechanical Engineering

School of Engineering

College of Engineering and Physical Sciences

University of Birmingham

January 2021

ABSTRACT

The aim of this research is to improve the ability of the Bees Algorithm to tackle global optimisation problems. The Bees Algorithm was formulated and inspired by the foraging behaviour of honeybees. The proposed enhancements target the initialisation and global search stages of the algorithm. The reason for this is that the initialisation stage could save efforts by directing the search earlier towards the more promising areas of the search space, leading to a better optimised result. Targeting during the global search is due to the researcher's belief that the neighbourhood search depends on it and any improvement will positively affect the neighbourhood search.

In this research, three enhancements were formulated based on the manipulation of the search space. The first enhancement (BAwSSR) involves continuous and gradual reduction of the search space with different scenarios that vary according to the starting point of reduction. The second enhancement (BADS) deals with the segmentation of search space into independent segments while using two sampling approaches to tackle a wide variety of problems. The third enhancement (BAOSS) also involves the segmentation of search space but divides it into independent segments to increase flexibility in handling a wider range of problems.

These proposed algorithms were tested on 24 benchmark functions with a broad range of characteristics. This test involves performance comparisons with the Quick Artificial Bee Colony (qABC) and the Standard Particle Swarm Optimisation 2011 (SPSO2011) algorithms. The obtained test data indicated noticeable improvements with an adequate level of stability over the original Bees Algorithm. The results were supported by the Mann–Whitney significance test, showing the improvements are statically significant for both accuracy and speed. Additionally, the proposed algorithms were tested on two engineering problems that included a comparison with a group of competitor algorithms. However, only the first proposed algorithm (BAwSSR) showed an obvious improvement. The other two algorithms (BADS) and (BAOSS) did not reveal any improvement.

ACKNOWLEDGEMENTS

First of All, I would like to prise and thank Allah SWT for all his blessings and for guiding me during the difficult time and giving me the strength to accomplish this research until it is completed.

My special thanks to, Professor Duc Truong Pham, my supervisor for giving me the opportunity to conduct the research and for his invaluable guidance and regular advice until the last minute of submitting this thesis.

A special gratitude to my research colleagues, Mr. Syahril Bahari, Mr. Shafie Kamaruddin for providing the knowledge and patiently answering my questions. I would like to also express my sincere thanks to all of my research colleagues who have been always supportive special my colleague Ismail Asrul which have been always ready to help.

To my parents who always initiated the hope and aspiration with their prayers. Special dedication to my mother, who throughout her life believed in me and assured me during the difficult circumstances.

To my wife for her endless encouragement and who patiently endured the stressful period

during my busy tim. To my children who enriched me with their warm love.

To my brothers and sisters who have been always proud of me and I cannot express enough my appreciation for their reassurance words.

This thesis was copy edited for conventions of language, spelling, and grammar by Paulina S. Cossette, Ph.D., at eContent Pro International.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF ABBREVIATIONS

ABC	Artificial Bee Colony
ACO	Ant Colony Optimisation
AS	Ant System
BA	Bees Algorithm
BADS	Bees Algorithm with Domain Segmentation
BAOSS	Bees Algorithm with Overlapping Search Space Segmentation
BAwSSR	Bees Algorithm w Search Space Reduction
BCO	Bee Colony Optimisation
BS	Bee System
CPU	Central Processing Unit
DE	Differential Evolution
DPSO	Discrete Particle Swamp Optimisation
EAS	Elitist Ant System
GA	Genetic Algorithm
GCPSO	Guaranteed Convergence Particle Swarm Optimiser
GVNS	General Variable Neighbourhood Search
HACO	Hybrid Ant Colony Optimisation
JSP	Job Shop Scheduling Problem
MMAS	Max–Min Ant System
MMO	Multimodal Optimisation
NFE	Number of Function Evaluations
NP	Non-deterministic Polynomial-time
OPF	Optimal Power Flow
OR	Operation Research
PSO	Particle Swarm Optimisation
QAP	Quadratic Assignment Problem
qABC	Quick Artificial Bee Colony

RAS	Rank Based Ant System
RVNS	Reduced Variable Neighbourhood Search
SA	Simulated Annealing
SI	Swarm Intelligence
SR	Success Rate
Std. Dev.	Standard Deviation
TS	Tabu Search
TSP	Travelling Salesman Problem
VRP	Vehicle Routing Problem
VNS	Variable Neighbourhood Search
VND	Variable Neighbourhood Decent

LIST OF SYMBOLS

c_1	Cognitive coefficient for Standard Particle Swarm Optimisation
c_2	Social coefficient for Standard Particle Swarm Optimisation
d	Number of dimensions
f	Objective or cost function(s)
K	Number of informants for Standard Particle Swarm Optimisation
l	Limit for abandonment in Quick Artificial Bee Colony
N	Number of decision variables
nb	Number of best sites in the Bees Algorithm
ne	Number of elite sites in the Bees Algorithm
ngh	Size of patches including site and its neighbourhood in the Bees Algorithm
nrb	Number of bees recruited for ($b-e$) sites in the Bees Algorithm
nre	Number of bees recruited for e sites in the Bees Algorithm
ns	Number of scout bees in the Bees Algorithm
p	Size of the bees' population in the Bees Algorithm
r	Neighbourhood radius for Quick Artificial Bee Colony
$rand$	Random vector element between 0 and 1 following the uniform distribution
S	Swarm size for Standard Particle Swarm Optimisation
$stlim$	Stagnation limit for the Bees Algorithm
T	Matrix transpose
x	Parameter to be optimised; design or decision variable(s); can be continuous, discrete, or a mixture of both
w	Inertia weight for Standard Particle Swarm Optimisation

Chapter 1 INTRODUCTION

1.1 Background

Optimisation is an everyday activity. In fact, it is a property of the human mind. For example, a person performing a task for the first time might face difficulties. However, after performing the task several times, the brain starts to discover better ways of performing these tasks. Moreover, at a higher level, for activities such as determining investments or shopping at a grocery store, the mind will naturally evaluate available options to find the best ones. Previous experiences are recalled or an analogy to the current situation is drawn, and strategies developed to solve problems that are regularly encountered.

As humanity began to modernise and industrialise, more complicated problems emerged. This is when the Operation Research discipline, which includes metaheuristics and exact methods, arose (Sorensen *et al.*, 2017). Researchers then began to formulate strategies to tackle such problems; one of the earliest efforts was G. Polya's book, *How to Solve It*, which introduces high-level strategies for solving some complicated problems (Sorensen *et al.*, 2017). However, for some problems, using exact methods is infeasible; this includes problems for which processing time increases exponentially as the size of the problem increases, meaning that they cannot be solved in polynomial time (or NP-hard problems). Some examples are the Travelling Salesman Problem (TSP), the Vehicle Routing Problem (VRP), and the Knapsack Problem (KP). This situation necessitated the use of stochastic techniques, which are usually employed in metaheuristic methods. Many metaheuristic methods have borrowed their inspiration from nature; therefore, they are referred to as nature-inspired algorithms.

For example, evolutionary algorithms, which are considered one of the earliest existing optimisation algorithms, are based on the theory of evolution, or what is commonly known as Darwinian theory. Evolutionary algorithms borrowed some principles of evolutionary theory, such as survival of the fittest and natural selection. Other algorithms are inspired by the social behaviour of some insects and animals. For example, the Ant Colony Optimisation algorithm (ACO) mimics the foraging behaviour of ants in their search for food. The Bees Algorithm (BA), Artificial Bee Colony (ABC), and Bee Colony Optimisation (BCO) are motivated by the foraging activity of Bees in nature. Others, like Particle swarm Optimisation (PSO), are inspired by the swarm of groups of birds. Metaheuristics also include algorithms based on natural phenomenon, such as the annealing of some physical material like steel and iron. Metaheuristics are applied successfully in many problems such as robotics, circuit design, radio signal processing, cloud computing, internet load and traffic balancing, and even vehicle fuel consumption.

1.2 Motivation

With their potential to solve complicated problems, the successful application of metaheuristics in many essential aspects of people’s lives has attracted significant attention. Hence, the development rate of new methods has increased, and existing algorithms with improved variations and hybridised versions have been released. These efforts aim to further improve the performance of these algorithms and to tackle some of the critical issues affecting their performance. The present research focuses on leveraging the performance of the BA by treating some of its weaknesses. One of these issues is the high degree of randomness and bias in initial and global search sampling due to the nature of randomly extracted samples and their uneven distribution across the search space. This lack of diversification and its effect on global search has been highlighted by Yuce *et al.*, (2017) and Packianather *et al.*, (2014). With the BA method of sampling, the process is highly likely to overlook certain areas of the search space where the optimum value might be located, causing the neighbourhood search to fail and to become trapped in local optima. The exploitative ability of the BA has been noted by Pham and Castellani (2015), who indicate its sensitivity to the absence of information about search direction. Moreover, the evenly extracted sample is likely to positively affect the search convergence speed with such informative sampling. Search convergence speed is another critical issue facing several metaheuristic algorithms. This research also addresses another BA vulnerability—slow convergence when optimising smooth unimodal functions (Pham *et al.*, 2008; Pham and Castellani, 2015).

1.3 Aim and Objectives

The general aim of this research is to improve the performance and optimisation capabilities of the BA in terms of accuracy and speed through search space manipulation of problems with continuous domains. To accomplish this, the following specific objectives have been set:

1. To develop an enhanced version of the BA with a gradual search space decrease during the initialisation and global search stages with different decreasing scenarios.

2. To develop a search space segmentation strategy with two types of sampling procedures. The first one is to takes each sample from one different segment. The second one to take every parameter from the same sample from different segment to handle different optimisation problem types of values. Additionally, a new procedure to sample from a mobile subset of the search space method will be used.
3. To develop a search space segmentation strategy with overlapping segments to handle different types of parameter values for optimisation problems.

1.4 Research Methods

The research methods used to achieve the above aims and objectives include:

- Surveying metaheuristic methods with a special focus on population-based, nature-inspired algorithms to figure out the current gaps and to identify performance issues.
- Reviewing research literature around the BA and its variants and applications to identify weaknesses and strengths to develop appropriate enhancements.
- Developing the proposed enhancements in R Studio using R programming.
- Assessing the proposed enhancements to performance on a wide variety of mathematically formulated benchmark functions with continuous domains.
- Evaluating the proposed algorithms on well-known, constrained, engineering problems.
- Conducting tests to identify the statistical significance of the improved performance of the proposed algorithms.

1.5 Thesis Outline

This thesis is organised as follows.

Chapter 2 reviews the literature on various optimisation metaheuristic methods with a special discussion of the concept of optimisation. It also discusses the main categories of metaheuristic methods, highlighting several, with special emphasis on population-based and nature-inspired methods. The chapter provides additional details of some of the widely adopted population-based methods, including their variants, hybridisation, and application. The weaknesses of these algorithms are discussed and analysed.

Chapter 3 presents the first proposed enhancement of the BA and the strategy of gradual decrease, with a discussion of its theoretical background. It also introduces the five implementations of the proposed strategy and lists the benchmark functions for testing the algorithm. The chapter also provides an analysis of the performance of the new BA on benchmark functions in terms of accuracy, speed, consistency and stability of the achieved result. The analysis includes a comparison with the original BA and two other algorithms widely used in the optimisation literature—ABC and PSO. A discussion of the analysis of the statistical significance of improvement is provided.

Chapter 4 introduces the second proposed enhancement of the BA and the strategy of search space segmentation for different sampling orientations to accommodate diverse types of parameter values. The chapter reviews the test results on benchmark functions compared with those for ABC and PSO. The results are analysed and discussed in terms of accuracy, speed, robustness, and reliability. Further, the results of the Mann–Whiney significance test are analysed. This is followed by a discussion of the statistical significance of the improvement.

Chapter 5 highlights the third suggested improvement of the BA, based on search space overlapping segmentation. The performance test results for benchmark functions are reviewed, and comparisons with ABC and PSO are discussed according to accuracy, speed, robustness, and reliability. Moreover, the outcome of the Mann–Whitney statistical significance test is reviewed.

Chapter 6 discusses the application of the three proposed BA enhancements on two well-known engineering problems. Both problems involve single objectives. The first, the Gear Train problem, is an unconstraint optimisation problem; the second, the tension/compression spring, is a constrained optimisation problem. The test results are compared using figures taken from the literature for the other algorithms examined.

Chapter 7 concludes this thesis, summarises the contributions of the research and suggests potential future work.

Chapter 2 LITERATURE REVIEW OF OPTIMISATION ALGORITHMS

2.1 Preliminaries

This chapter surveys some of the popular techniques used in optimisation and their different applications. A classification of these techniques is presented based on their stochastic features. Although these techniques include swarm and non-swarm intelligence methods, the survey focuses more on swarm intelligence methods for continuous domain problems.

2.2 Optimisation

Optimisation is concerned with finding the best solution possible with available resources (Chinneck, 2000). Here, “best” means the fittest solution in the search space, although there is no guarantee that a global solution exists. Mathematically, optimisation is defined as the use of innovative strategies to find a set of values that minimises an objective function. This definition also applies to maximisation without loss of generality. This can be achieved by inverting the sign of the objective function. Mathematically, optimisation can be formulated as follows:

$$\min_{\chi \in \mathbb{R}^{\mathrm{N}}} f(x),\ \chi = \left(\chi_{1,}\chi_{2,} \cdots \chi_{\mathrm{N}}\right) \qquad (2.1)$$

where:

f = the objective (cost) function(s), defined in $\mathbb{R}^{\mathrm{N}}$, which is the search space or search solution defined in the set of real numbers; and

X = the parameters or the design variables of the objective functions; these parameters can be discrete, continuous, or a mix of both.

2.3 Classification of Optimisation Algorithms

Optimisation techniques are widely applied in a large variety of research subjects, such as math and physics, as well as business and decision-making processes. Currently, there exists a plethora of optimisation algorithms. However, according to the "no free lunch" theorem of Wolpert and Macready (1997), there is no algorithm that can solve all kinds of problems.

Optimisation algorithms can be separated into two main categories: deterministic and stochastic. Deterministic algorithms are often applied when the problem to optimise is not too complicated or the dimensionality of the problem is not too high, rendering the optimisation process infeasible or time consuming (Weise, 2009). Examples of these types of algorithms are branch and bound, state space search, and algebraic geometry. On the other hand, stochastic algorithms are concerned with the types of problems considered to have high complexity, according to the computational theory of complexity, such as NP-hard and NP-complete (Abidin *et al.*, 2011). The heavy consumption of resources required to solve these problems makes a deterministic approach infeasible, which is where metaheuristics comes into play (Alia and Mandava, 2011; Glover, 2006; Bianchi *et al.*, 2009). The metaheuristics method is a group of algorithms that aims to find optimal or near-optimal solutions within a designated polynomial time and with available resources. However, metaheuristics does not guarantee that the exact global optimum solution will be reached, nor does it provide a universal algorithm that can solve all kinds of problems.

2.4 Metaheuristics

Metaheuristics can be classified into two main groups: single-solution-based and population-based algorithms. Single-solution-based algorithms, also known as trajectory methods (Boussaïd *et al.*, 2013), can find solutions following a trajectory pathway in a search space (Baghel *et al.*, 2012).

2.4.1 Single-Solution-Based Metaheuristics

Single-solution-based metaheuristics initially create one individual solution and gradually improve it. Tabu search, simulated annealing, iterated local search, and variable neighbourhood search are examples of this type of algorithm.

2.4.1.1 Tabu Search

The Tabu Search (TS) algorithm uses memory as an element to store a search history in a list of solutions. The aim is to prevent the search from endlessly revisiting the same search area (Boussaïd *et al.*, 2013), which is usually a characteristic of being trapped in local optima. This enforces more explorative behaviour in the search for the optimum. The type of memory used can vary from short-term memory to intermediate and long-term memory (Boussaïd *et al.*, 2013; Glover, 1990), which affects the algorithm's explorative traits. Because TS was devised by Glover in 1978, there were many attempts to hybridise it with the BA (Shafia *et al.*, 2011; Imanguliyev, 2013) and (ACO; Eswaramurthy *et al.*, 2009). TS works better with discrete search spaces than with continuous spaces because it needs to visit the exact value stored in the list; this is possible when there is a limited number of values, as in a discrete domain. This is particularly difficult in a continuous domain, where the search space can be divided infinitely, making it extremely large, especially with high-dimensionality problems. Hence, in a continuous domain, TS becomes inefficient with the extremely large search space, which will eventually make the list grow massively and become exceedingly costly in terms of processing power and time. Moreover, the existence of an infinite number of values will cause the search to become inefficient, as it is highly unlikely that one value will be searched again (Luke, 2013).

2.4.1.2 Simulated Annealing

Simulated annealing (SA), which appeared in the early 1990s, is one of the earliest algorithms devised (Kirkpatrick *et al.*, 1983). It is a popular single-solution-based algorithm derived from the process of heating and cooling of metallurgical materials. To achieve the desired properties for the material, such as hardness, flexibility, and ductility, the heating process requires starting with a higher temperature and gradually decreasing it. The temperature affects the atomic movement of the material, which is more random at higher temperatures. As the temperature cools, the random movement of the atoms decreases, and they are frozen and linked with strong bonds (Nolle *et al.*, 2011). The critical factor is the gradual cooling of the temperature, governed by the Boltzmann probability factor, which enables control of the material's desired attributes. An analogous concept was adopted in the SA algorithm. That algorithm works by generating an initial solution **S**; if, in the following iteration, a better—or at least similar—solution S^* is generated, the new solution will be considered the current solution, and the search will continue. If the generated solution is worse than the current one, it will be accepted with probability $e^{\frac{\triangle}{T}}$, where $\triangle = f(S) - f(S^*)$, and T is a factor corresponding to the temperature in the annealing process. SA was applied effectively to many continuous and discrete optimisation problems (Boussaïd *et al.*, 2013). It has also been applied successfully to machine learning and neural networks to escape local minima (Owen and Abunawass, 1993). Figure 2.1 shows the pseudo code of SA.

```
1    Select initial solution S randomly.
2    Select initial temperature T.
3    While stopping criteria not met, repeat.
4        Generate $S^*$.
5        If $f(S) \geq f(S^*)$, then
6            $S \leftarrow S^*$
7        else
8            $S \leftarrow S^*$ when $e^{\frac{\triangle}{T}}$ > random (0,1).
9        End.
10       Reduced T.
11   If stopping criteria not met, continue.
12   End.
```

Figure 2.1 Pseudo code of simulated annealing algorithm (SA)

2.4.1.3 Iterated Local Search

Iterated local search (ILS) is an improved version of hill-climbing algorithms (Luke, 2013) with frequent random restarts. It is a single-solution-based algorithm. The main idea behind ILS is to generate a random solution; it will then select a point in the vicinity of the current local optimum to find a better solution. This happens by a perturbation in the current local optimum (Boussaïd *et al.*, 2013). The point should not be too far or too close to the current local optimum. It keeps moving from one local optimum to another within the search space during the consecutive iterations (Luke, 2013). However, the solution will need to be verified by acceptance criteria that control the balance between diversification and intensification (Boussaïd *et al.*, 2013). Many combinatorial optimisation problems have been successfully optimised with ILS (Lourenço *et al.*, 2003). Some examples of these problems are the TSP and the job scheduling (JS) problem, including a wide variety of problem settings, from single-machine to complex-multimachine scheduling. Figure 2.2 includes the pseudo codes for ILS.

```
1    Generate initial solution randomly $S$.
2    Using local search, generate S* from S.
3    Repeat.
4    Get solution P from S* via perturbation.
5    Using local search, generate P* from P.
6    Apply acceptance criteria.
7    If P* satisfy the acceptance criteria, then
8            S* ← $P*$
9    End.
10   Until the stopping criterion is met.
11   End.
```

Figure 2.2 Pseudo code for ILS

2.4.1.4 Variable Neighbourhood Search

Mladenović and Hansen (1997) established the Variable Neighbourhood Search (VNS). It suggests that the search around a randomly generated solution should be within a dynamically changing neighbourhood area. First, the structures of the neighbourhood are randomly initialised: $\mathrm{N}_1 \mathrm{N}_2 \ldots . \mathrm{N}_{n\,max}$. The next step is to generate an initial solution s that is followed by the initiation of the VNS main cycle, where s' is arbitrarily selected from the nth neighbourhood of the current solution s. A method of local searching is then used on s' to generate s''. If s'' is better than s, then s'' will replace the current solution s. The search then continues with the next neighbourhood structure, N_2, restarting the cycle. The pseudocode of VNS is shown in Figure 2.3. However, some solutions might be searched many times due to overlapping neighbourhood structures that affect search efficacy (Boussaïd *et al.*, 2013; Battiti *et al.*, 2008). VNS was primarily used for combinatorial problems and later for problems with continuous domains, using a Gaussian distribution to generate noise. Many variations of VNS have been produced, such as the deterministic version of variable neighbourhood descent (VND). Reduced VNS (RVNS) is another variation in which random points of neighbourhood structure $\mathrm{N}_k(x)$ are selected. It is somewhat similar to the Monte Carlo method but with more controlled randomisation. Many other versions of VNS exist, such as Skewed VNS (SVNS) and general VNS (GVNS). In SVNS, the search tries to go further from the incompetent solutions, whereas in GNS, VNS itself will be embedded in a local search. Historically, VNS has been applied to a wide variety of problems, such as vehicle routing problems (VRP), single and parallel scheduling problems, time tabling, and the Knapsack problem (Hansen *et al.*, 2008).

```
1   Generate neighbourhood structure $N_n, n = 1,2,\dots n_{max}$.
2   Randomly select initial solution s.
3   Repeat.
4   $n \leftarrow 1$
5   While $n < max$ do
6   Arbitrarily choose solution $s'$ from the $n$th neighbourhood of the $N_n(s)$ s.
7   Using local search, generate $s''$ from $s'$.
8   If $s''$ is better than $s'$,
9   s $\leftarrow s''$
10  $n \leftarrow 1$
11  else
12  $n \leftarrow n + 1$
13  End.
14  End.
15  Until the stopping criterion is met.
```

Figure 2.3 Pseudo code of VNS

2.4.2 Population-Based Metaheuristics

Rather than relying on a single solution, population-based metaheuristics generate a collection of candidate solutions. The fact that population-based methods are expected to provide a better quality of solution, or at least to converge faster, is intuitive, given that more solutions will be tried in every iteration during the search process. Population-based methods have a long history of borrowing concepts from nature; this is how a nature-inspired algorithm came into existence. As noted above, methods like evolutionary algorithms were inspired by Darwinian theory. However, algorithms such as ACO, ABC, and BA are based on what is known as swarm intelligence (SI). SI describes a form of intelligence derived from the behaviour of social insects living in swarms, such as ants, bees, birds, and animal herds (Blondin, 2009). The concept of SI emerged from the group collective intelligence and self-organised behaviour of simple entities operating collectively within a decentralised system for the whole group's benefit (Bonabeau *et al.*, 1999). This system has certain characteristics, such as self-organisation, homogeneous membership, internal communication, decentralised decision-making, and allocation of tasks. Self-organisation is supposed to be the outcome of decision-making and the allocation of tasks. Decision-making requires a form of local communication that could be happening directly or indirectly (Yang *et al.*, 2018). The allocation takes place without direct commands to individuals (Gordon, 1996). However, task allocation does not happen arbitrarily; it involves a division of labour in which every group performs specialised tasks. This could be easily observed in bees when each swarm of bees in the hive is assigned a specialised task. For example, scout bees are responsible for finding and collecting information about new flower patches and must inform other groups of worker bees, called foragers, who are responsible for collecting nectar from those flower patches. In an ant colony, one group of ants is responsible for building the nest, while others, called a task force, oversee the protection of the colony (Seeley, 2002). This communication involves an exchange of information through performing different dances or by exchanging signals (Anderson and Ratnieks, 1999), a crucial property of SI.

2.4.2.1 Genetic Algorithm

The genetic algorithm (GA) is one of the most popular optimisation algorithms. It was first established by John Holland and his students in 1975 (Whitley, 1994). As mentioned before, the GA was inspired by the principles of survival of the fittest and natural selection.

Based on these concepts, the fittest individuals of a population are most likely to survive to the next generation. These individuals are called chromosomes. Chromosomes in biology consist of genes that correspond in the GA literature to the decision variables sampled from the search space. To achieve the survival of the fittest concept, individuals are subject to selection, crossover, and mutation operations introduced in the GA to encourage evolutionary growth to produce enhanced offspring. First, the selection operator is used to select individuals with the highest fitness to be mated via crossover operators. To locate the fittest of chromosomes, the fitness function is applied. To enhance the selected chromosomes, the crossover operator is applied to exchange and recombine genes from selected chromosomes. This random recombination could take place at one point of the gene, or at two points or more (Davis, 1991; Maini *et al.*, 1994). The chromosomes are encoded either as binaries or as real-number schemes. A real-number scheme is more appealing for the continuous domain, whereas a binary scheme fits into a combinatorial domain (Herrera *et al.*, 1998). The aim is to guarantee that the best individuals are moved to the next generation. Thus, in the case that the evolutionary operators have yielded no better individuals, the selected parents themselves are moved to the next generation.

Finally, the mutation operator is applied; here, small parts of the chromosome are perturbed to ensure local diversity of the improved result by the crossover operator. Mutation effects are akin to neighbourhood search in the BA, where it is perturbing the existing solution to generate an improved one. However, a crossover job is more of an exploratory search within the dimensions of the initial sample (Qi and Palmieri, 1994). This limits the GA's ability to explore the search space evenly and to produce a diversity of solutions. There have been many variations of GA, such as the elitism GA, the steady-state

GA, generation gap methods, and GA-with-a-tree-style genetic programming (Luke, 2013). These variations include attempts to hybridise GA with other algorithms, as in hill climbing (Luke, 2013), ABC (Kumar and Kumar, 2017), and PSO (Hyma *et al.*, 2010). GAs have also been applied to a broad spectrum of applications, such as circuit design, robotics, pattern recognition, and biology (i.e., to study the immune system). Additionally, the GA was applied in software testing (Aljahdali *et al.*, 2010) and flow shop scheduling (Murata and Ishibuchi, 1994) with acceptable performance.

2.4.2.2 Particle Swarm Optimisation

In 1995, PSO was proposed by Eberhart and Kennedy (1995) as a simple optimisation algorithm. This SI algorithm simulates the behaviour of a flock of birds or a school of fish living and travelling in groups where individuals move in harmony in their search for food. The particles in the PSO represent the candidate solution. The simulation of swarms of birds moving freely in space corresponds to the candidate solutions changing their position in the search space. Every particle has its own velocity and position, and the particles continuously update their position and velocity according to neighbouring particles and their previous experience. PSO is one of the most popular algorithms and is commonly used in real-life optimisation problems; this is due to its easy integration with other algorithms and to its simplicity, which makes it easy to use, even for non-expert researchers (Resende *et al.*, 2018). Moreover, PSO can benefit from modern computer technology because it has no consecutive stages and can be executed in parallel (Resende *et al.*, 2018).

As with many other widely used algorithms, there are some weaknesses that need to be addressed. One weakness in PSO when solving multimodal problems is premature convergence (Liang *et al.*, 2006; Resende *et al.*, 2018). When the search space is too complex, having many local optima, PSO can easily become stuck in one of these local optima when particle velocity is restricted to update only from the best global position. This restriction is usually implemented to force PSO to converge faster. To solve this problem, Liang *et al.* (2006) suggested using the comprehensive PSO in which the velocity is updated according to the history of the best velocity of all particles. One of the earliest improvements of PSO was the velocity clamping PSO (Resende *et al.*, 2018), which was suggested to tackle a known PSO phenomenon called the swarm explosion effect, where the velocity factor in the PSO increases arbitrarily at an extremely high rate. Intuitively, the proposed solution was to stipulate a maximum value that velocity could not exceed. Van den Bergh and Engelbrecht (2002) suggested the guaranteed convergence particle swarm optimiser (GCPSO) as one of the variants of PSO that was proposed to tackle its inability to converge in certain cases. Originally, PSO was suggested for optimising problems with continuous domains and later for discrete PSO (DPSO); however, DPSO was proposed to address problems with discrete domains, such as JS problems, vehicle routing, and the TSP (Kennedy and Eberhart, 1997).

Laskari *et al.* (2002) also attempted to address discrete problems by proposing three PSO variants: PSO-In, PSO-Bo, and PSO-Co. According to Laskari, these variants were tested on seven test problems with remarkable success, proving the ability of PSO to handle integer-optimisation problems. Over time, PSO has been widely hybridised with some well-known algorithms, such as ABC, to optimise neural networks (Wang *et al.*, 2015). It has also been hybridised with GA (Gandelli *et al.*, 2005) to produce a more effective algorithm called GSO. However, PSO has been applied to a wide variety of real-life problems, such as neural networks (Niu *et al.*, 2007; Wang *et al*., 2015). Other applications of PSO have been in image classification (Omran, 2004) and to determine the ideal location of gas and oil wells (Onwunalu and Durlofsky, 2010).

2.4.2.3 Ant Colony Optimisation

ACO is another SI algorithm inspired by ant swarm foragers in their search for food. It appeared as a proposal by M. Dorigo and his colleagues (Dorigo and Caro, 1999; Dorigo *et al.*, 2006). Originally, the ant system (AS) was suggested to solve combinatorial optimisation problems and was applied to solve the TSP problem, the quadratic assignment problem (QAP), and the JS problem (Dorigo *et al.*, 1991; Dorigo *et al.*, 1996). Consequently, ACO was proposed by Dorigo *et al*. (1996) as a new optimisation algorithm. The concept of ACO is that, during their search for food, ants naturally secrete a substance called a pheromone to enable follower ants to recognise the path to the food. Likewise, the follower ants secrete pheromones as they pass along that path, enforcing the existing pheromones and attracting more ants to follow (Shtovba, 2005). However, if the food source is too far or if it starts to decrease and is finally exhausted, the pheromones will evaporate because fewer ants will follow, until consequently no further pheromones are deposited, and the path is eventually abandoned.

With ACO gaining more attention, many variants and hybrids have been produced, such as Max–Min Ant System (MMAS), Elitist Ant System (EAS), and Rank-Based Ant System (RAS), which address the TSP and handle the problem of being trapped in local optima (Chaparro and Valdez, 2013; Prakasam and Savarimuthu, 2016). Another ACO variant has been developed for the continuous optimisation domain to handle the protein–ligand docking problem and to predict the protein–ligand structure (Korb *et al*., 2007). Additionally, ACO was hybridised with the Fuzzy C-means technique to produce the hybrid ACO (HACO) algorithm, which was applied to classify power signal disturbance patterns (Biswal *et al.*, 2011). According to Biswal *et al.* (2011), HACO is capable of classifying signal disturbance patterns. Further, ACO has been hybridised with the biogeography-based optimisation (BBO) technique (Savsani *et al.*, 2014) to improve performance. Generally, ACO has performed well in the combinatorial field, especially for problems like telecommunication.

2.4.2.4 Artificial Bee Colony

ABC is an optimisation method inspired by the behaviour of bees foraging for honey. It was proposed by Dervis Karaboga in 2005 (Karaboga 2005; Karaboga and Basturk, 2007). Initially, ABC was suggested for continuous domain problems and it was adapted later for the combinatorial domain where a new version for discrete optimisation applied to job shop scheduling (DABC) was released (Pan *et al*., 2010; Thammano and Phu-ang, 2013). Many attempts have been made to introduce new variants to improve ABC. One of these attempts, by Zhu and Kwong (2010), was to incorporate global-best-guided ABC (GABC), which was inspired by the global-best concept in PSO. The aim was to improve ABC's exploitation ability.

The main concept of ABC is based on dividing the bee colony into three groups: employed bees, onlooker bees, and scout bees. The bees in the colony are divided equally as employed bees and onlooker bees. Employed bees are placed arbitrarily on food sources, with each bee associated with one food source. Every food source corresponds to a fitness solution. Employed bees return to the hive and share the information about the food source with the onlooker bees after attempting to improve it within the neighbourhood size. The selection of the food source to recruit onlooker bees will depend on its individual fitness relative to the fitness of the overall food sources found so far. If food sources become exhausted, the employed bees will begin serving as scout bees, and they will start the search for new food. Food will be considered exhausted after a certain number of iterations without improvements.

Although the ABC algorithm is one of the most competitive, it suffers from some weaknesses, specifically, boor exploitation (Zhu and Kwong, 2010). To tackle this problem, Zhu and Kwong (2010) suggested borrowing the g-best concept from PSO. Another ABC weakness, as reported by Qiu *et al.* (2013), is the problem of getting stuck in local optima at an early stage of the search and converging exceptionally slowly to the optimum. However, this common problem in stochastic optimisation algorithms is due to the lack of balance between exploration and exploitation (Santos and Alotto, 2011). Santos and Alotto (2011) suggested introducing the Gaussian distribution to generate candidate solutions.

Nevertheless, ABC has gained great popularity and was hybridised with some other popular algorithms, such as PSO and GA. For instance, ABC was combined with PSO to benefit from the direct exchange of information about the global solution. This was essential to improve performance in terms of exploration and exploitation (Kıran and Gündüz, 2013). Another attempt was to hybridise ABC with GA to improve the large tuning parameters of the FOFP-FOPID 2-DOF robotic system (Kumar and Kumar, 2017). To improve ABC further, its author, Dervis Karaboga, released quicker ABC (qABC), in which he introduced critical alterations to the original algorithm (Karaboga and Gorkemli, 2012). In the original form, only three parameters existed: colony size, the limit of trials before food will be abandoned, and the number of cycles (i.e., the maximum number of iterations). However, with qABC, a new parameter ‘r’ was introduced, which represents the neighbourhood radius for the Euclidean distance from the selected solution position for more intensified and exploitative search. This updates the solution position of onlooker bees with the aim of improving the overall performance of ABC. Additionally, a new limit calculation was introduced, which considers the dimensions of the problems in hand. The author reported significant improvement by qABC. In general, ABC has performed remarkably in many applications. When applied to automatically generate software testing cases, satisfactory performance was also reported by the researcher (Dahiya *et al.*, 2010). Finally, ABC was used in power systems to optimise the performance of fault section estimation and performed effectively (Huang and Liu, 2013).

2.4.2.5 Bees Algorithm

BA, developed by Pham *et al.* (2005), is one of the major contributors in the field of metaheuristic algorithms. As mentioned previously, BA was designed based on the foraging behaviour of bees in nature. It begins with the initialisation stage when several scout bees (n) are arbitrarily distributed in the search area. The next stage—the local search or neighbourhood search—is when the bees are recruited for the neighbourhood search around the best locations (m) within a certain distance factor (ngh) according to the ranking created for the initial sample using the fitness function. The elite bees (e) are selected from the fittest m that was previously selected. More bees are recruited to search around the elite bees (e), while fewer bees are recruited to search around the non-elite (m−e). The remaining scout bees (n−m) are assigned to search randomly in the search space, which takes place in the global search. The overall bees, which consist of (e + (m−e) + (n−m)), constitute the new population that will be used in the next iterations to continue the search. The search continues until the stopping criterion is met. The stopping criterion is usually either reaching a certain number of iterations or finding a satisfactory value for the optimum (or near the optimum). The following is a list of BA parameters that need to be initialised by the user:

i. number of scout bees (*ns*),

ii. number of best sites (*nb*) out of sites visited by *ns*,

iii. number of elite sites out of *nb* selected sites (*ne*),

iv. number of bees recruited for *ne* sites (*nre*),

v. number of bees recruited for the other *nb-ne* selected sites (*nrb*), and

vi. size of neighbourhood (*ngh*).

The stopping criterion can be either a predefined maximum function evaluation or finding the optimum defined with a stipulated threshold. The steps for the BA in its basic form are:

```
1. Initialise the scout population with random solutions.
2. Evaluate fitness of the population.
3. While (stopping criterion not met) //Forming the new population.
4. Select sites for the neighbourhood search.
5. Recruit bees for selected sites (more bees for elite sites) and evaluate fitness.
6. Select the fittest bee from each patch.
7. Assign remaining bees to search randomly and evaluate their fitness.
8. End While.
```

The BA, like many other population-based metaheuristics, experiences some problems due to its stochastic nature. These problems include being trapped in local optima in functions such as Rosenbrock, Langerman, and Bukin6, and the slow convergence to the optimum due to the lack of search guidance. Although the BA exhibits reliable performance in noisy problems, it shows weaknesses when optimising smooth unimodal functions like Rosenbrock (Hansen *et al.*, 2009) where it experiences slow performance (Pham *et al.*, 2008b, 2008a). While the BA sample generation depends greatly on randomness, it contains a greater number of control parameters that need to be tuned, which implies that the process of parameter tuning by itself is an optimisation problem.

From its inception in 2005, the significance of the BA as a metaheuristic method has grown constantly (Kamsani, 2016) and it has been subject to many improvements and hybridisations. One of the attempts proposed by the original developer of the BA, Pham *et al.* (2008c), incorporates the neighbourhood shrinking and the abandonment procedures into the basic BA (BBA). The shrinking procedure is applied if a solution does not yield any results that improve upon what has been achieved so far; in so doing, the search neighbourhood area around that solution is decreased by 80%. However, if a solution has been searched more than a certain limit, that solution will be abandoned and a new one will be generated to replace it.

Hussein *et al.* (2014) also attempted to improve the initialisation phase of the BA through incorporating levy flight in the neighbourhood search. Levy flight takes the shape of a random walk with varying lengths of steps, from short to long, and the researchers reported significant improvement over the BA (Hussein *et al.*, 2014). Another attempt, by

Shatnawi *et al.* (2013), introduced the BA with two types of memory-based lists (local and global); here, the researchers claim a minimum of 59% improvement over the BA. In a subsequent attempt, Yuce *et al.* (2013) proposed the use of adaptive neighbourhood search that shrinks and enlarges according to fitness values for both the shrinking and abandonment procedures. The findings confirmed that the proposed method performed significantly better than the BA, and it was particularly better in high-dimension problems (Yuce *et al*., 2013).

Although the BA was originally proposed for continuous domains, a new BA version for combinatorial domains has been formulated. It was used, first, to solve the single machine scheduling problem (Pham *et al*., 2007). Subsequently, an attempt to enhance the BA was made using the TRIZ methodology to optimise printed circuit board (PCB) assemblies (Mei *et al*., 2010). Meanwhile, Ozbakir *et al.* (2010) attempted to solve the generalised assignment problem (GAP). The authors assumed the outcome of the experiment exhibited the capability of the BA to solve larger GAP problems.

Furthermore, the BA has been applied to a wide variety of applications. For example, in the field of electrical and electronic engineering (EEE), it was applied to the optimal power flow (OPF) model (Anantasate *et al*., 2010), with the researchers reporting productive results and the capacity to apply the BA to any size of OPF. Another application of the BA in the EEE field is to power systems (Satheesh, 2013) through the hybridisation of the BA with neural networks; this indicates the ability of the hybrid method to preserve system stability. An important application of the BA was to improve power–torque distribution in hybrid electric vehicles (Derakhshan and Shirazi, 2014). The results reported for the experiment showed improved control performance along with a reduction of fuel consumption and pollution.

Another field where the BA has been applied is in computer science and engineering. It was used in software testing with an acceptable result, although not the best among all methods involved in the performance evaluation (Zabil and Zamli, 2013). Yang *et al.* (2015) applied the BA in wireless communications and signal recognition. The researchers used a hybridised version of the BA with a neural network (Yang *et al.*, 2015). The field of robotics has also attracted researchers' attention. One attempt to use the BA was by Pham *et al.* (2009), who maintained two-link acrobatic robot (ACROBOT) balance and stability using a fuzzy logic controller. According to the researchers, the BA exhibited good performance in tuning the fuzzy logic controller. Another attempt in robotics, by Eldukhri and Kamil (2015), involved the tuning of the robot gymnast (Robogymnast) swing-up control parameters. The results indicated stability and a significant decrease in swing-up time (Eldukhri and Kamil, 2015).

2.5 Summary

In this chapter, a survey of wide variety of metaheuristic methods has been conducted, giving more consideration to population-based and nature-inspired algorithms. The survey included details about when these algorithms were created, their variance, and their applications. Additionally, a review to some weaknesses of these methods is presented. A common problem recognised among these algorithms is keeping a good balance between their two main aspects: exploration and exploitation. If an algorithm performs exploration excessively, this might result in premature convergence to the local optima. If the algorithm is too explorative, it might result in a slow convergence to the solution. However, the BA has some of its unique weaknesses, which are highlighted in this chapter. Briefly, these problems are related to the number of parameters that need to be set up before initiating the algorithm. Moreover, a reduction of the degree of randomness could help to improve performance. This creates room for further improvement of the BA. In the following chapters, new techniques will be proposed to tackle these problems and to improve overall performance.

Chapter 3 THE BEES ALGORITHM WITH SEARCH SPACE REDUCTION (BAwSSR)

3.1 Preliminaries

Although metaheuristics follow an approximate approach to find an optimal solution, many of these algorithms experience slow convergence rates, getting trapped in local optima and long computational times, particularly for hard problems (Liang, 2006; Beheshti and Shamsuddin, 2013). This can be attributed to the stochastic nature of these algorithms and the high number of iterations they must perform. BA is not an exception (Pham and Darwish, 2010; Alfi and Khosravi, 2012; Yuce *et al.*, 2015; Kamsani, 2016). This chapter presents a new method for improving the BA, targeting the initialisation and the global search stages. The interest in focusing on these two stages is based on the researcher's belief that the neighbourhood search stage largely depends on what the initialisation and global search stages are providing to it. A neighbourhood search refines what has been given to it. The proposed technique is based on applying the gradual reduction of search space in the initialisation and global search stages.

The review of the literature of BA improvements indicates that most of the attempts to use search space manipulation have targeted the neighbourhood search phase. One such attempt, proposed by Ghanbarzadeh (2007), aims to increase the exploitability of the neighbourhood search area with a large patch size parameter. In this process, the BA neighbourhood search parameter, ngh, is initially selected with a relatively wide size, and a reduction in size follows if no promising patches are discovered. Unlike Ghanbarzadeh's attempt, the proposed technique by Azfanizam (2014) involves increasing the size of the neighbourhood search area around the selected patch if no improvement is achieved; here, the goal is to expand the radius of the neighbourhood area, assuming that this will speed up the convergence to the optimal solution (Azfanizam, 2014). Additionally, the author assumes that this proposal helps us to escape the local optima (Azfanizam, 2014). To the best of this researcher's knowledge, no attempt has been made to apply the concept of search space reduction in the global search or initialisation stages. Furthermore, the literature on the BA reveals that the initialisation and global search stages have not been given enough consideration in terms of enhancements of the BA (Hussein *et al.*, 2014).

The proposed method is inspired by two numerical optimisation methods, namely, bracketing and region elimination. It employs an adapted notion of the region elimination concept to achieve abandonment and reduction of search space within the BA. Additionally, to make global searches more intelligent, the roles of bees for searching in the global search stage have been varied, with different swarms of bees performing their searches in different parts of the search area.

3.2 Bracketing-Region Elimination Method

Bracketing methods comprise two approaches: exhaustive search and bounding phase methods (Deb, 2012). The algorithm proposed here is derived from bracketing–exhaustive search in addition to the region elimination method. The exhaustive search involves exploring the whole search space to find the optima at equally located intervals (Nievergelt *et al.*, 1995; Deb, 2012).

In the region elimination method, the core concept is to consecutively eliminate some parts of the search space until the exact minimum is found. The steps of the region elimination method are as follows:

1. A search space region will be specified, for example (a, b), where $a < b$.
2. Two points will be selected, x_1 and x_2, where $x_1 < x_2$.
3. $f(x_1)$ will be evaluated, and
    - If $f(x_1) < f(x_2)$, the minimum cannot exist beyond x_2 in the period (x_2, b), and the segment (x_2, b) is abandoned from the search space.
    - If $f(x_1) > f(x_2)$, the minimum cannot exist before x_1 in the period (a, x_1), therefore, the segment (a, x_1) is abandoned from the search space (Bhattacharjya, 2009). An illustration is provided in Figure 3.1.

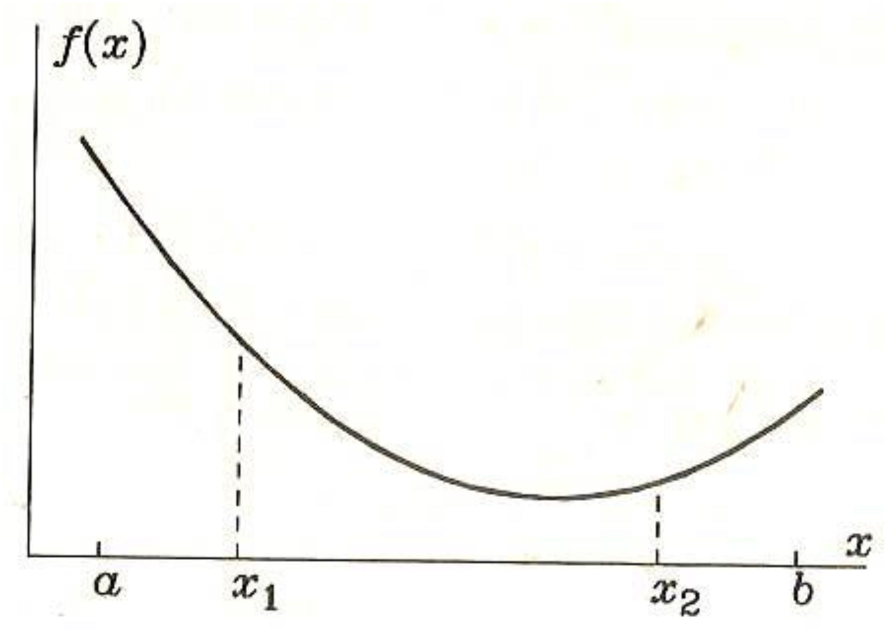


Figure 3.1 Region elimination method

3.3 The BA with Search Space Reduction (BAwSSR)

To adapt the region elimination method for this proposed algorithm, a few fundamental changes should be applied. The region elimination method was primarily designed for optimising unimodal functions; however, the proposed method should cater to multimodal problems where there are more than one local and/or global optimum. Another fundamental change is that, while the region elimination method is a deterministic optimisation approach used to find the exact minimum, the minimum or near-minimum will satisfy the requirements for the BA as a metaheuristic approximation method. Hence, the following modifications to the region elimination method were introduced:

- The search space region (a, b) is considered as the whole search domain specified for the problem being tested, where $a < b$, and $L = (b-a)$
- The number of points selected should be equal to the initial sample parameter n of the BA (x_1, x_2, … x_n).
- The elimination interval is noted as $S = \Delta x = (b-a)/n$, where n is the number of initial samples.
- F(x), {$X = (x_1, x_2, \dots x_n)$ is evaluated, and
    a. If ($f(x)$–$f_{\text{optm}} > 0.001$, the segments (b , $b-\Delta x$) and (a , $a+\Delta x$) will be eliminated from the search space. The new search space should include only the segment ($b-\Delta x$, $a+\Delta x$), and the process should continue.
    b. If ($f(x)$–$f_{\text{optm}} \leq 0.001$, the search will be terminated.
- If the search does not yield values close to the optimum within the specified error value, this process is restarted from the original region (a, b), and the

search for the optimum is repeated until the maximum number of function evaluations (NFEs) is reached (500,000).

Furthermore, to better use the global search, bees are assigned to search in different areas of the search space. There are five search scenarios for the search space:

- Searching the whole search space, which decreases gradually from both ends of the search space by the elimination factor $S = (b-a)/n$ (Figure 3.2).
- Searching after the first quarter from the two ends of the search space, which gradually decreases from both ends by the elimination factor S (Figure 3.3).
- Searching the area between the centre and the left end of the search space (a), which gradually decreases from both sides by the elimination factor S (Figure 3.4).
- Searching only the area between the centre and the right end of the search space (b), which gradually decreases from both sides by the elimination factor S (Figure 3.5).
- Searching the whole search space, which gradually decreases from the left end (a) by elimination factor S (Figure 3.6).

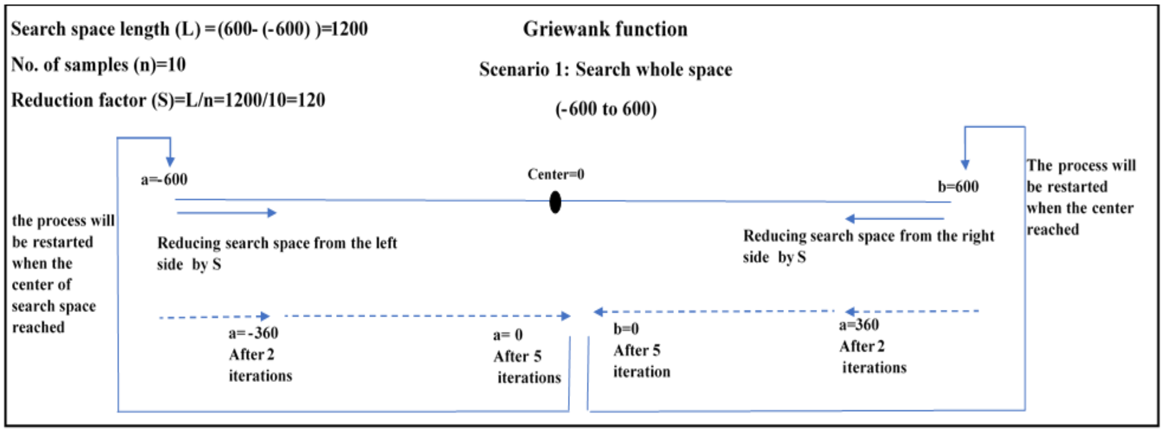

Search space length (L) =(600- (-600))=1200
No. of samples (n)=10
Reduction factor (S)=L/n=1200/10=120
Griewank function
Scenario 1: Search whole space
(-600 to 600)
a=-600
Center=0
b=600
The process will be restarted when the center reached
the process will be restarted when the center of search space reached
Reducing search space from the left side by S
Reducing search space from the right side by S
a=-360 After 2 iterations
a= 0 After 5 iterations
b=0 After 5 iteration
a=360 After 2 iterations

Figure 3.1 Search scenario 1

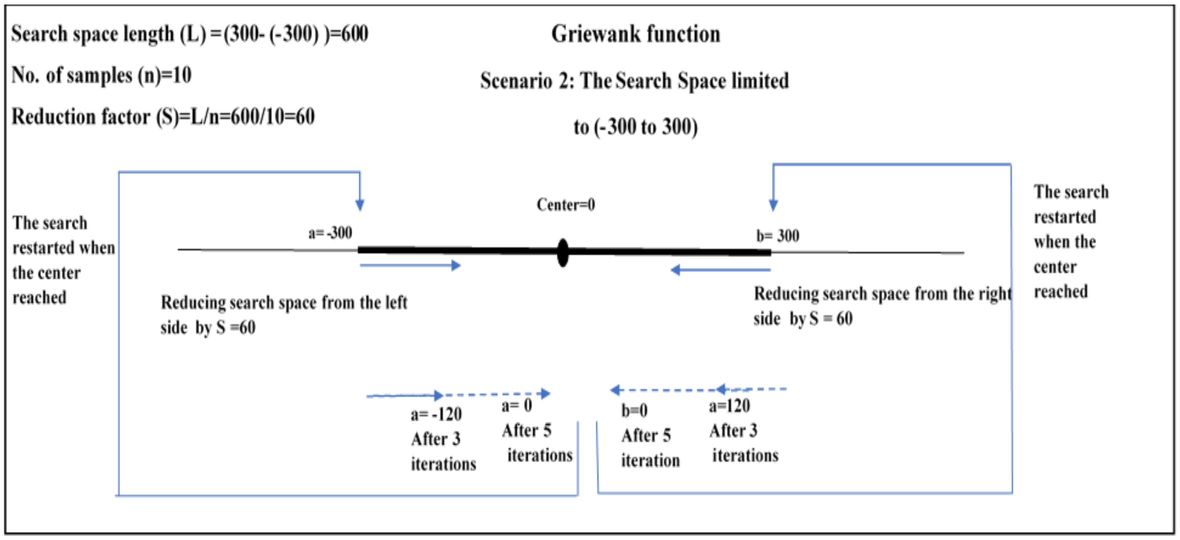

Search space length (L) =(300- (-300))=600
No. of samples (n)=10
Reduction factor (S)=L/n=600/10=60
Griewank function
Scenario 2: The Search Space limited
to (-300 to 300)
The search restarted when the center reached
a= -300
Center=0
b= 300
The search restarted when the center reached
Reducing search space from the left side by S =60
Reducing search space from the right side by S = 60
a= -120
After 3 iterations
a= 0
After 5 iterations
b=0
After 5 iteration
a=120
After 3 iterations

Figure 3.2 Search scenario 2

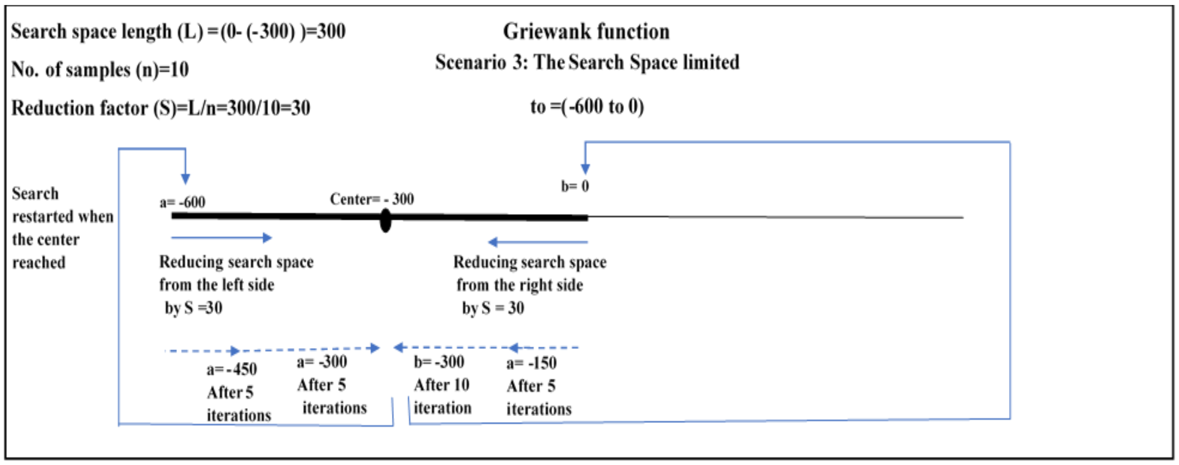

Search space length (L) =(0- (-300))=300
No. of samples (n)=10
Reduction factor (S)=L/n=300/10=30
Griewank function
Scenario 3: The Search Space limited
to =(-600 to 0)
Search restarted when the center reached
a= -600
Center= - 300
b= 0
Reducing search space from the left side by S =30
Reducing search space from the right side by S = 30
a= -450
After 5 iterations
a= -300
After 5 iterations
b= -300
After 10 iteration
a= -150
After 5 iterations

Figure 3.3 Search scenario 3

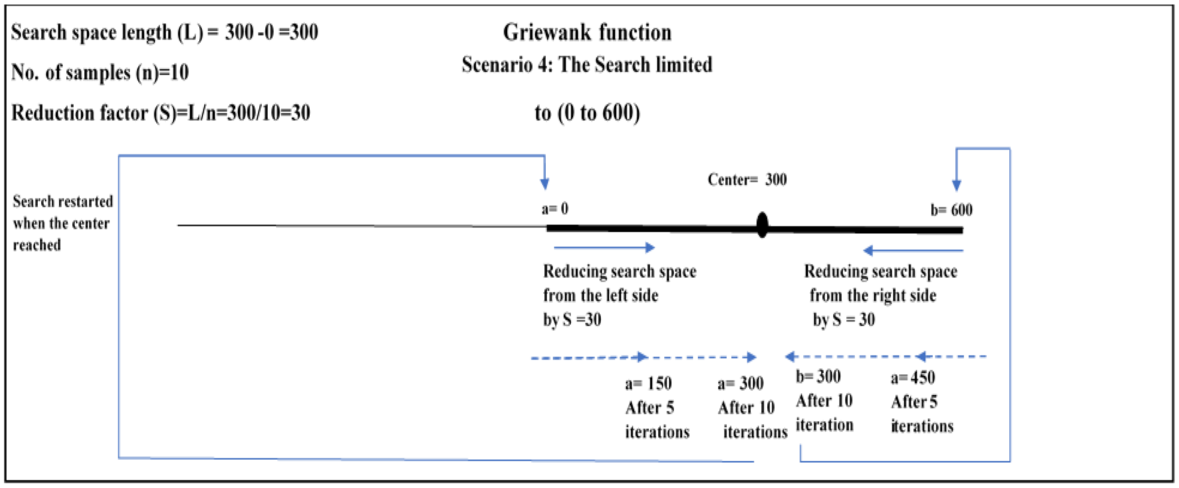

Search space length (L) = 300 -0 =300
No. of samples (n)=10
Reduction factor (S)=L/n=300/10=30
Griewank function
Scenario 4: The Search limited
to (0 to 600)
Search restarted when the center reached
Center= 300
a= 0
b= 600
Reducing search space from the left side by S =30
Reducing search space from the right side by S = 30
a= 150 After 5 iterations
a= 300 After 10 iterations
b= 300 After 10 iteration
a= 450 After 5 iterations

Figure 3.4 Search scenario 4

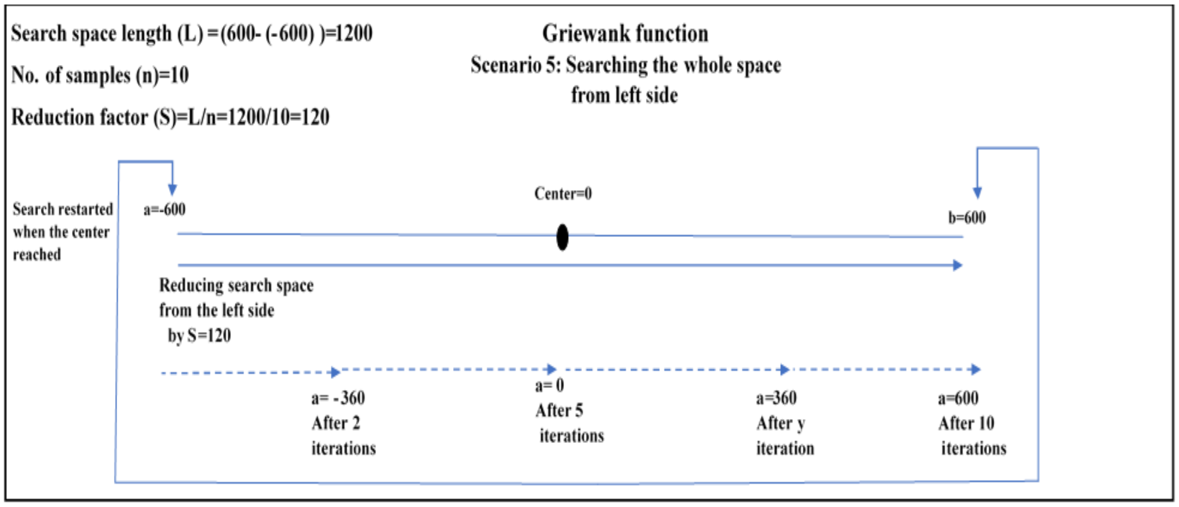

Search space length (L) =(600- (-600))=1200
No. of samples (n)=10
Reduction factor (S)=L/n=1200/10=120
Griewank function
Scenario 5: Searching the whole space
from left side
Search restarted when the center reached
a=-600
Center=0
b=600
Reducing search space from the left side by S=120
a= -360 After 2 iterations
a= 0 After 5 iterations
a=360 After y iteration
a=600 After 10 iterations

Figure 3.5 Search scenario 5

The pseudocode for the BAwSSR algorithm is:

- Initialise the population with random solutions using the search space reduction technique.
- Evaluate the fitness of the population.
- While (stopping criterion not met), forming new bee population.
- Select best and elite bees for neighbourhood search.
- Recruit bees around selected sites and evaluate fitness.

- Select the fittest bee from each site.
- Assign remaining bees to search randomly using the search space reduction technique with the five scenarios and evaluate their fitness.
- If (stopping criterion not met), reduce the search space by *L*.
- End while.

The algorithm requires several control parameters that the user must determine:

1. Number of scout bees (*ns*),
2. Number of best sites (*nb*) out of sites visited by *ns,*
3. Number of elite sites out of *nb* selected sites (*ne*),
4. Number of bees recruited for *ne* sites (*nre*),
5. Number of bees recruited for the other *nb–ne* selected sites (*nrb*), and
6. Size of neighbourhood (*ngh*).

3.4 Experiment Setup

One well-known method of measuring optimisation algorithm performance is to use test functions, or what are commonly known as benchmark functions. These functions represent complex mathematical problems used to challenge the performance of optimisation tools. Because many of these functions represent real-world problems, they have been used widely as performance indicators in a variety of scientific disciplines, such as manufacturing, physics, and economics (Imanguliyev, 2013; Kamsani, 2016). For example, the Sphere and Rosenbrock functions, respectively, represent real-world cost curve and cost minimisation problems (Imanguliyev, 2013; Rosenbrock, 1960). However, the Ackley functions relate to the representation of the surfaces of some material particles, such as protein (Imanguliyev, 2013; Dieterich and Hartke, 2012).

The criteria for selecting benchmark functions for testing were designed to account for the varying degrees of search space topography, complexity, separability, and modality. This was essential for ensuring testing objectivity and reliability (Jamil and Yang, 2013). Complexity includes separability and dimensionality. Separability refers to the interdependency between function parameters; non-separable functions are typically harder to solve than are separable functions. The dimensionality of a function demonstrates its number of decision variables, or what are sometimes called parameters. As the number of dimensions in a function increases, it becomes harder to optimise. Modality is a property related to the number of peaks in the search space; the function is unimodal if it has only one global optimum, otherwise, with many global and/or local optima, it is regarded as multimodal (Jamil and Yang, 2013). The 24 test functions used in this research are listed and classified according to their properties in Table 3.1. Although more than half of these are two-dimensional, they are predominantly multimodal and non-separable, which adds to the difficulty of solving them.

Table 3.1 List of test functions and their properties

Functions	Differentiability	Separability	Scalability	Modality
Sphere (10D)	Yes	No	Yes	Unimodal
Rosenbrock (10D)	Yes	No	Yes	Unimodal
Quartic (30D)	Yes	Yes	Yes	Unimodal
Ackley (10D)	Yes	No	Yes	Multimodal
Schaffer (2D)	Yes	No	Yes	Multimodal
Easom (2D)	Yes	Yes	No	Multimodal
Rastrigin (10D)	Yes	Yes	Yes	Multimodal
Shekel (4D)	Yes	Yes	Yes	Multimodal
Langerman (10D)	Yes	No	Yes	Multimodal
Griewank (10D)	Yes	No	Yes	Multimodal
Branin (2D)	Yes	No	No	Multimodal
Sumpow (10D)	Yes	Yes	Yes	Unimodal

Bukin6 (2D)	No	No	No	Multimodal
Crossit (2D)	No	No	No	Multimodal
Drop (2D)	No	Yes	No	Multimodal
Shubert (2D)	Yes	Yes	No	Multimodal
Beale (2D)	Yes	No	No	Multimodal
McCorm (2D)	Yes	No	No	Multimodal
Camel6 (2D)	Yes	No	No	Multimodal
Boha1 (2D)	Yes	No	No	Multimodal
Colville (2D)	Yes	No	No	Multimodal
Powersum (2D)	Yes			Unimodal
Salomon (2D)	Yes	No	Yes	Multimodal
Alpine (2D)	Yes	Yes	No	Multimodal

The test designed for this study investigated the performance of the proposed algorithm. The three key performance metrics used in this investigation were: the fitness value, the success rate (SR), and the NFE. The fitness value refers to the accuracy of the optimum found when compared to the standard. It can also be described as the quality of the solution. The SR refers to the number of times the algorithm was able to converge to the optimum within the maximum NFEs permitted in all the runs. The NFE denotes the number of times the benchmark functions used in testing were executed for every individual run. It also can be described as the speed with which the algorithm converged with the optimum. This is clear, as the algorithm needs less time to converge if the function under testing is executed only a few times. The approach of this research was to preliminarily consider the quality of the solution or its accuracy, and then to look at the SR and the function evaluation. Hence, this research should contribute to the quality of the solution. The results of the test are compared to those of the BBA and two other well-known and popular algorithms, PSO and ABC. As discussed in Chapter 1, there are many versions of PSO. However, SPSO2011 was selected primarily because it is one of the latest versions of PSO, in addition to its dimensionality features and its remarkable performance against separable and unimodal functions (Zambrano-Bigiarini *et al.*, 2013). On the other hand, the qABC version of ABC is one of the latest versions of the ABC algorithm and the quick version of ABC (Kamsani, 2016); this means that it could converge rapidly to the optimum, which makes it attractive for testing.

The algorithm should run until the stopping criteria are met. The stopping criteria are defined as follows:

- Either the global optimum is found with an acceptable error rate (ER; chosen here as $ER < 0.001$), or
- The maximum NFEs is reached (stipulated in this research as 500,000).

Fifty independent runs were conducted for every function. The results obtained from these runs were then analysed using the mean and standard deviation. The mean, standard deviation, and quality of the solution together reflect the reliability and robustness of the solution achieved (Shanghooshabad and Abadeh, 2016). This is because the algorithm, stuck in the same local optima in every run, might generate small means and standard deviations even though performance is poor. These statistics were additionally collected from the BBA, the SPSO2011, and the qABC for performance comparison. Finally, a statistical significance test was conducted to demonstrate the significance of the proposed algorithm's performance over that of the other algorithms used in the comparisons. This was calculated using the Mann–Whitney test. This test was selected because it is used when the figures obtained are without specific distribution or when it is not possible to predict how the obtained variables will be dispersed throughout the search space (Nachar, 2008). This makes it more appropriate for the research, given the stochastic aspect of the algorithms involved. The final decision about performance was made according to the statistical significance of the accuracy value obtained as well as the NFE's.

3.5 Results and Discussion

3.5.1 Solution Quality (Accuracy)

To assess solution quality, the test was conducted according to the parameter settings listed in Tables 3.2 and 3.3 for BAwSSR and BBA, respectively. The test was also carried out to compare the proposed algorithm's performance with the qABC and SPSO2011 using the parameter settings in Tables 3.4 and 3.5, respectively. Table 3.6 contains the results comparing the performance of the BAwSSR and BBA algorithms in terms of accuracy. It reveals that the proposed algorithm, BAwSSR, performed better than the BBA in all 24 functions. Table 3.6 also presents the results that compare BAwSSR's performance against that of the SPSO2011 and qABC. The findings indicate an improvement by BAwSSR. Specifically, it outperformed SPSO2011 and qABC in 22 and 20 functions, respectively. The above findings indicate BAwSSR was able to find more accurate optimum than other algorithms; nevertheless, to assess stability and consistency, the acquired figures need to be analysed using statistical measures like the mean and standard deviation.

Table 3.2 List of parameter values used for testing BAwSSR

No.	Functions	n	m	nsp	e	nep	ngh	stlim
1	Sphere (10D)	11	4	10	2	30	0.003	10
2	Rosenbrock (10D)	11	4	10	2	30	0.003	10
3	Quartic (30D)	13	4	10	2	30	0.005	10
4	Ackley (10D)	11	4	10	2	30	0.00001	10
5	Schaffer (2D)	11	4	10	2	30	0.01	10
6	Easom (2D)	100	4	10	2	30	0.0009	10
7	Rastrigin (10D)	11	4	10	2	30	0.003	10
8	Shekel (4D)	1,000	4	10	2	30	0.001	10

9	Langerman (10D)	1,000	4	10	2	30	0.09	10
10	Griewank (10D)	17	10	10	2	30	0.001	10
11	Branin (2D)	15	10	10	2	30	0.01	10
12	Sumpow (10D)	20	4	10	2	30	0.0001	10
13	Bukin6 (2D)	10	4	10	2	30	0.00001	10
14	Crossit (2D)	15	4	10	2	30	0.05	10
15	Drop (2D)	11	4	10	2	30	0.01	10
16	Shubert (2D)	1,000	4	10	2	30	0.0005	10
17	Beale (2D)	45	4	10	2	30	0.045	10
18	McCorm (2D)	10	4	10	2	30	0.05	10
19	Camel6 (2D)	100	4	10	2	30	0.005	10
20	Boha1 (2D)	11	4	10	2	30	0.05	10
21	Colville (2D)	10	4	10	2	30	0.01	10
22	Powersum (2D)	10	4	10	2	30	0.01	10
23	Solomon (2D)	10	4	10	2	30	0.001	10
24	Alpine (2D)	13	4	10	2	30	0.001	10

Table 3.3 List of parameter values used for testing BBA

No.	Functions	n	m	nsp	e	nep	ngh	stlim
1	Sphere (10D)	11	4	10	2	30	0.003	10
2	Rosenbrock (10D)	10	4	10	2	30	0.003	10
3	Quartic (30D)	13	4	10	2	30	0.005	10
4	Ackley (10D)	10	4	10	2	30	0.01	10
5	Schaffer (2D)	10	4	10	2	30	0.01	10
6	Easom (2D)	10	4	10	2	30	0.01	10
7	Rastrigin (10D)	10	4	10	2	30	0.03	10
8	Shekel (4D)	10	4	10	2	30	0.01	10
9	Langerman (10D)	10	4	10	2	30	0.09	10
10	Griewank (10D)	10	10	10	2	30	0.1	10
11	Branin (2D)	10	10	10	2	30	0.001	10
12	Sumpow (10D)	10	4	10	2	30	0.1	10
13	Bukin6 (2D)	10	4	10	2	30	0.0001	10
14	Crossit (2D)	10	4	10	2	30	0.1	10
15	Drop (2D)	10	4	10	2	30	0.01	10
16	Shubert (2D)	10	4	10	2	30	0.1	10
17	Beale (2D)	10	4	10	2	30	0.05	10
18	McCorm (2D)	10	4	10	2	30	0.5	10
19	Camel6 (2D)	10	4	10	2	30	0.05	10
20	Boha1 (2D)	10	4	10	2	30	0.05	10

21	Colville (2D)	10	4	10	2	30	0.0005	10
22	Powersum (2D)	10	4	10	2	30	0.1	10
23	Solomon (2D)	10	4	10	2	30	0.01	10
24	Alpine (2D)	13	4	10	2	30	0.001	10

Table 3.4 qABC parameter settings

Parameter	Value
Population size	10
Cycles (max number)	100
Employed bees ne	5
Onlooker bees ne	4
Random scouts	1
Stagnation limit for site (abandonment stlim)	200
ra	1

Table 3.5 SPSO2011 parameter settings

Parameter	Value
Population size	100
PSO cycles (max number) T	500,000
Connectivity	Default:3
Maximum velocity	Default:1.1
C_1	Default:1.1
C_2	200
W_{max}	Default:0.7
W_{min}	Default:0.7

Table 3.6 Best performance figures for BAwSSR, BBA, qABC, and SPSO2011 for accuracy values

No.	Functions	Result of 50 runs		Result of 50 runs		Result of 50 runs	
		BAwSSR	qABC	BAwSSR	SPSO2011	BAwSSR	BBA
1	Sphere (10D)	50	0	50	0	50	0
2	Rosenbrock (10D)	50	0	41	9	50	0
3	Quartic (30D)	50	0	49	1	50	0
4	Ackley (10D)	50	0	50	0	50	0
5	Schaffer (2D)	50	0	50	0	50	0
6	Easom (2D)	19	31	50	0	32	18
7	Rastrigin (10D)	50	0	50	0	50	0
8	Shekel (4D)	20	30	50	0	34	16
9	Langerman (10D)	36	14	50	0	27	23
10	Griewank (10D)	50	0	50	0	50	0
11	Branin (2D)	19	31	0	50	39	11
12	Sumpow (10D)	37	13	38	12	45	5
13	Bukin6 (2D)	42	8	48	2	38	12

14	Crossit (2D)	31	19	50	0	28	22
15	Drop (2D)	50	0	50	0	50	0
16	Shubert (2D)	14	36	50	0	46	4
17	Beale (2D)	34	16	28	22	33	17
18	McCorm (2D)	28	22	50	0	29	21
19	Camel6 (2D)	28	22	50	0	38	12
20	Boha1 (2D)	50	0	50	0	50	0
21	Colville (2D)	47	3	25	25	38	12
22	Powersum (2D)	48	2	39	11	28	22
23	Solomon (2D)	50	0	50	0	50	0
24	Alpine (2D)	50	0	50	0	50	0
Total		BAwSSR: 20 qABC: 4		BAwSSR: 22 SPSO2011: 1 No winner: 1		BAwSSR: 24 BBA: 0	

Figures 3.7–3.15 below exhibit the performance in terms of accuracy by all the algorithms involved.

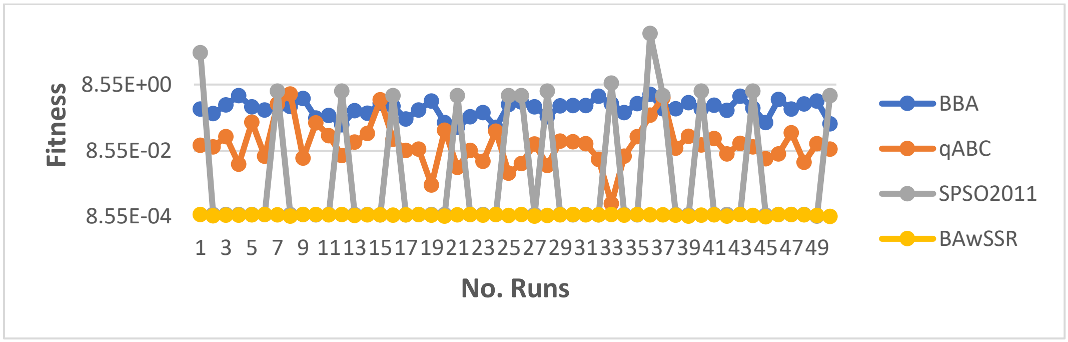


Figure 3.7 Result of 50 runs for BAA, BAwSSR, qABC, and SPSO2011 on Rosenbrock 10D function

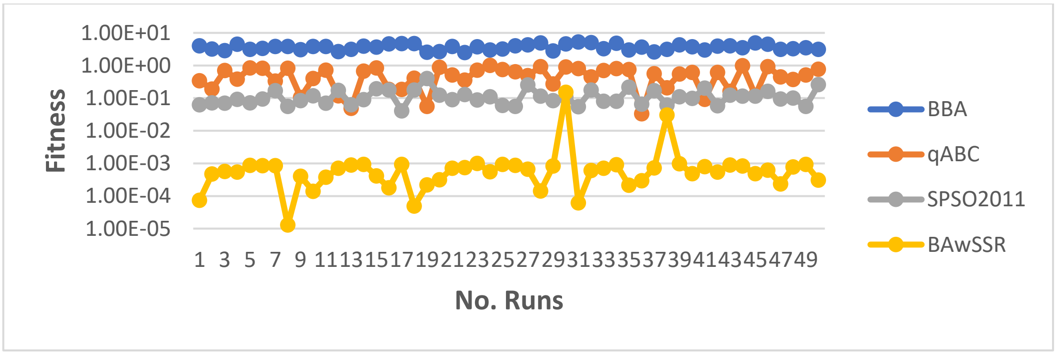


Figure 3.6 Result of 50 runs for BAA, BAwSSR, qABC, and SPSO2011 on Quartic 30D function

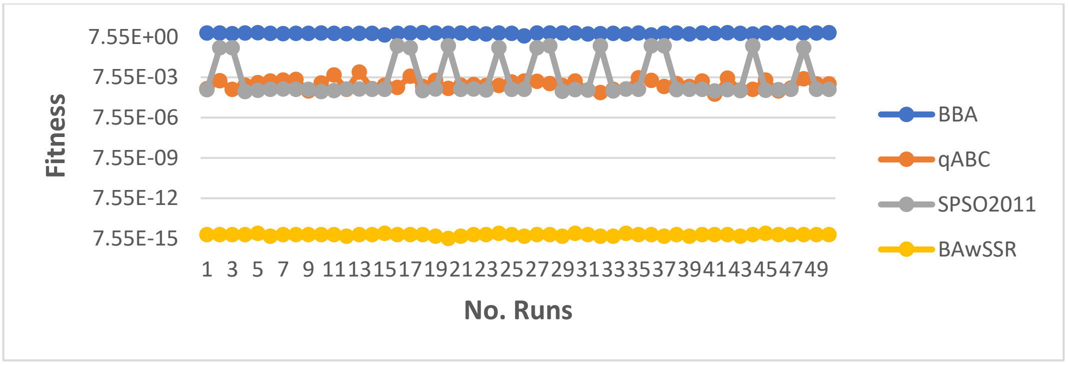


Figure 3.7 Result of 50 runs for BAA, BAwSSR, qABC, and SPSO2011 on Ackley 10D function

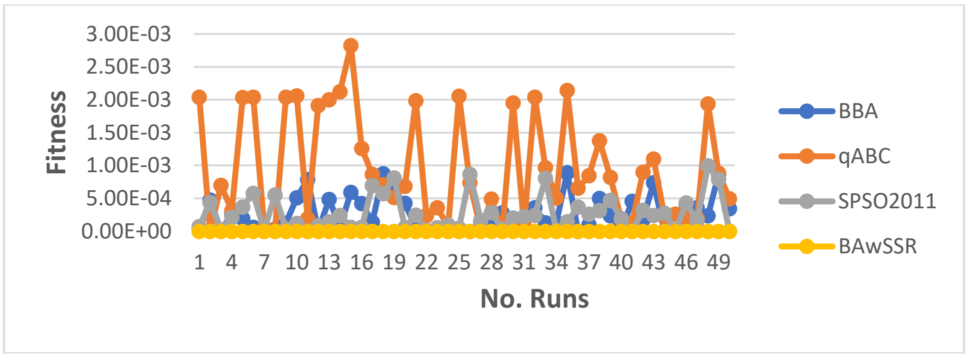


Figure 3.8 Result of 50 runs for BAA, BAwSSR, qABC, and SPSO2011 on Schaffer's 2D function

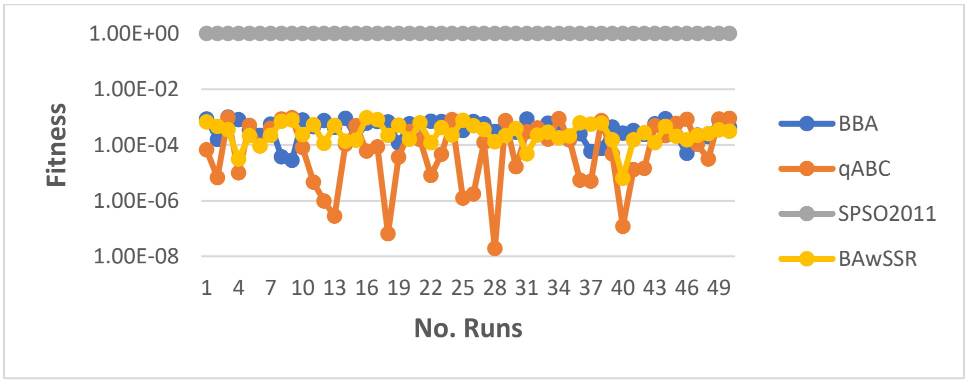


Figure 3.11 Result of 50 runs for BAA, BAwSSR, qABC, and SPSO2011 on Easom 2D function

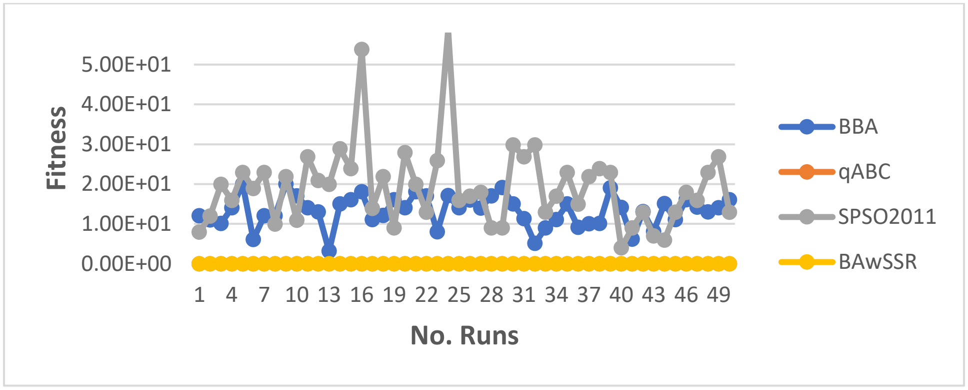


Figure 3.12 Result of 50 runs for BAA, BAwSSR, qABC, and SPSO2011 on Rastrigin 10D function

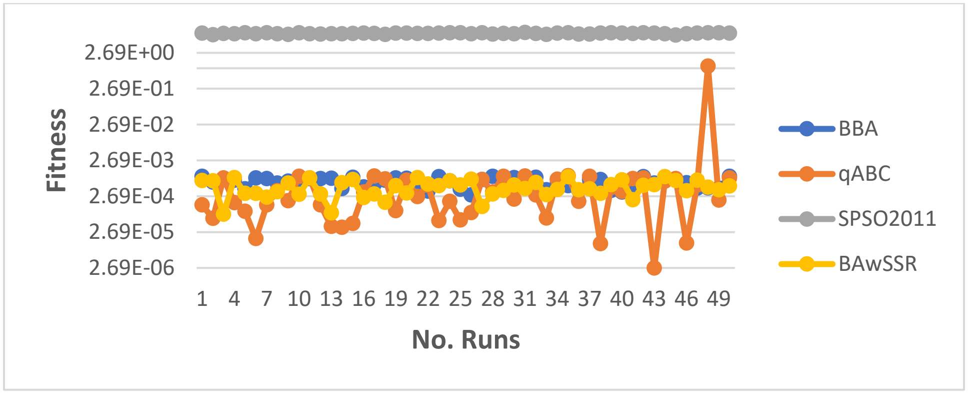


Figure 3.13 Result of 50 runs for BAA, BAwSSR, qABC, and SPSO2011 on Shekel 4D function

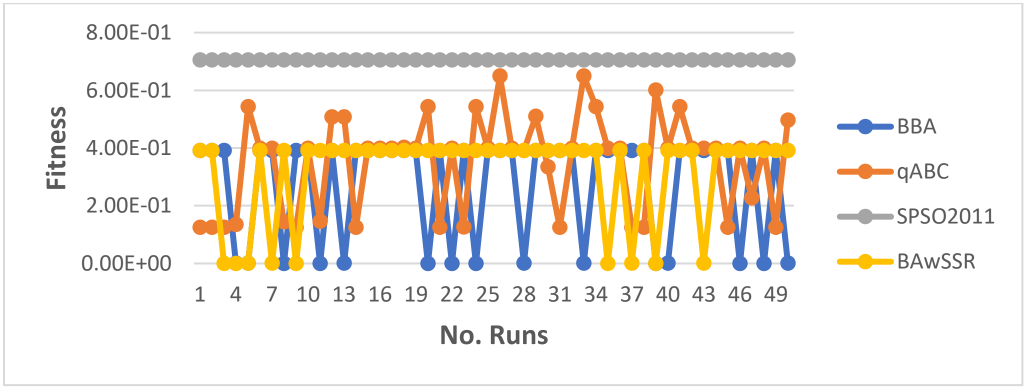


Figure 3.14 Result of 50 runs for BAA, BAwSSR, qABC, and SPSO2011 on Langerman 10D function

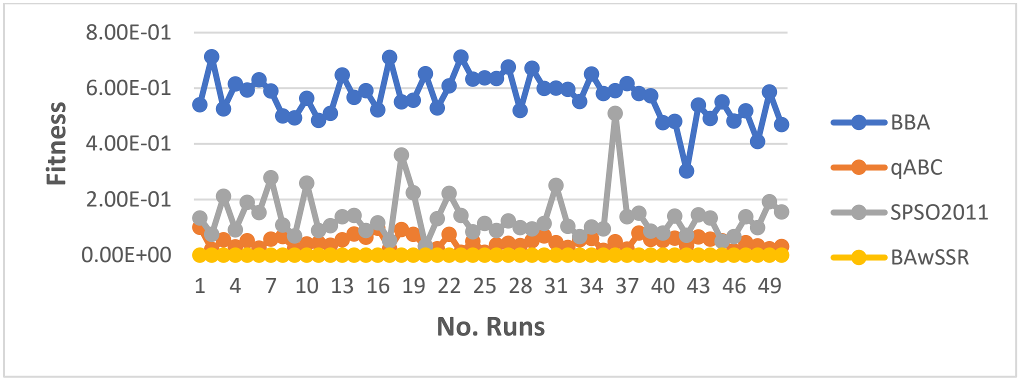


Figure 3.15 Result of 50 runs for BAA, BAwSSR, qABC, and SPSO2011 on Griewank 10D function

When examining the figures for the means and standard deviations in Table 3.7, captured for 50 runs, the data indicate that BAwSSR had lower means in 23 functions. However, lower than the above for the standard deviations showing that BAwSSR it is spreading closer to the mean in only 19 functions. In general, these results support the above conclusion about the improved performance of BAwSSR although not behaving fully consistent. The values in Table 3.8, which compare BAwSSR against qABC, also confirm the good performance, with BAwSSR achieving better means and standard deviations in 21 and 22 functions, respectively. Similarly, the findings presented in Table 3.9 for the comparison between BAwSSR and SPSO2011 support the enhanced performance, with BAwSSR exceling in 22 and 20 functions for the mean and standard deviation, respectively. Furthermore, the proposed algorithm was able to find the exact minimum in the following functions: Shaffer, Rastrigin, Griewank, and Drop. Additionally, the analysis of the figures demonstrates that, among all the algorithms examined, BAwSSR was enhanced in 18 functions, more than half of which were multimodal, non-separable, scalable, and differentiable. The overall conclusion is that BAwSSR was able to find more accurate optimum than the competitor algorithms while exhibiting good level of consistency.

However, to investigate whether BAwSSR's performance against the other algorithms was statistically significant, the Mann–Whitney test was conducted.

Table 3.7 Mean and standard deviation of best accuracy values for BAwSSR and BBA obtained through 50 independent runs on test functions f1–f24

No.	Functions	BBA		BAwSSR	
		Mean	Std. Dev.	Mean	Std. Dev.
1	Sphere (10D)	8.19E -04	1.27E-04	4.10E-32	1.31E-32
2	Rosenbrock (10D)	1.80E+00	9.23E-01	9.55E-04	3.72E-05
3	Quartic (30D)	3.70E+00	7.34E-01	4.18E-03	2.14E-02
4	Ackley (10D)	1.38E+01	1.25E+00	1.42E-14	2.25E-15
5	Schaffer (2D)	2.57E-04	2.61E-04	0.00E+00	0.00E+00
6	Easom (2D)	4.56E-04	2.64E-04	3.41E-04	2.29E-04
7	Rastrigin (10D)	1.33E+01	3.84E+00	0.00E+00	0.00E+00
8	Shekel (4D)	6.67E-04	1.96E-04	5.01E-04	2.26E-04
9	Langerman (10D)	2.74E-01	1.79E-01	3.21E-01	1.50E-01
10	Griewank (10D)	5.68E-01	7.78E-02	0.00E+00	0.00E+00
11	Branin (2D)	6.43E-04	1.96E-04	3.85E-04	2.95E-04
12	Sumpow (10D)	6.86E-04	2.44E-04	2.17E-04	3.05E-04
13	Bukin6 (2D)	1.59E-02	4.93E-03	1.02E-02	4.07E-03
14	Crossit (2D)	4.23E-04	2.71E-04	3.77E-04	2.57E-04
15	Drop (2D)	4.06E-04	2.48E-04	0.00E+00	0.00E+00
16	Shubert (2D)	5.24E-03	4.89E-03	4.93E-04	2.60E-04
17	Beale (2D)	4.82E-04	2.74E-04	4.21E-04	2.72E-04
18	McCorm (2D)	5.26E-04	2.92E-04	4.68E-04	2.60E-04
19	Camel6 (2D)	4.87E-04	2.86E-04	2.63E-04	2.94E-04
20	Boha1 (2D)	4.68E-04	2.92E-04	0.00E+00	0.00E+00
21	Colville (2D)	9.28E-04	7.65E-05	8.45E-04	1.45E-04
22	Powersum (4D)	6.84E-04	2.26E-04	6.42E-04	2.10E-04
23	Solomon (2D)	5.49E-04	2.57E-04	1.06E-15	3.07E-16
24	Alpine (2D)	6.25E-04	2.41E-04	1.07E-16	3.93E-17
Total		1	5	23	19

Table 3.8 Mean, and standard deviation of best accuracy values obtained through 50 independent runs on test functions f1–f24 for BAwSSR and qABC

No.	Functions	qABC		BAwSSR	
		Mean	Std. Dev.	Mean	Std. Dev.
1	Sphere (10D)	6.24E-04	2.49E-04	4.10E-32	1.31E-32
2	Rosenbrock (10D)	4.09E-01	8.85E-01	9.55E-04	3.72E-05
3	Quartic (30D)	5.23E-01	2.88E-01	4.18E-03	2.14E-02
4	Ackley (10D)	3.11E-03	3.05E-03	1.42E-14	2.25E-15
5	Schaffer (2D)	9.65E-04	8.04E-04	0.00E+00	0.00E+00
6	Easom (2D)	2.73E-04	3.28E-04	3.41E-04	2.29E-04
7	Rastrigin (10D)	6.22E-04	2.78E-04	0.00E+00	0.00E+00
8	Shekel (4D)	2.34E-02	1.61E-01	5.01E-04	2.26E-04

9	Langerman (10D)	3.50E-01	1.63E-01	3.21E-01	1.50E-01
10	Griewank (10D)	4.65E-02	2.19E-02	0.00E+00	0.00E+00
11	Branin (2D)	2.63E-04	2.76E-04	3.85E-04	2.95E-04
12	Sumpow (10D)	3.82E-04	1.12E-04	2.17E-04	3.05E-04
13	Bukin6 (2D)	2.81E-02	1.61E-02	1.02E-02	4.07E-03
14	Crossit (2D)	5.65E-04	3.01E-04	3.77E-04	2.57E-04
15	Drop (2D)	1.12E-02	2.17E-02	0.00E+00	0.00E+00
16	Shubert (2D)	2.69E-04	3.13E-04	4.93E-04	2.60E-04
17	Beale (2D)	1.98E-03	3.49E-03	4.21E-04	2.72E-04
18	McCorm (2D)	5.06E-04	2.83E-04	4.68E-04	2.60E-04
19	Camel6 (2D)	4.01E-04	3.26E-04	2.63E-04	2.94E-04
20	Boha1 (2D)	3.20E-04	2.78E-04	0.00E+00	0.00E+00
21	Colville (2D)	3.43E-02	3.10E-02	8.45E-04	1.45E-04
22	Powersum (4D)	7.13E-03	8.58E-03	6.42E-04	2.10E-04
23	Solomon (2D)	4.80E-02	4.88E-02	1.06E-15	3.07E-16
24	Alpine (2D)	5.58E-04	2.84E-04	1.07E-16	3.93E-17
Total		3	2	21	22

Table 3.9 Mean, and standard deviation of best accuracy values obtained through 50 independent runs on test functions f1–f24 for BAwSSR and SPSO2011

No.	Functions	SPSO2011		BAwSSR	
		Mean	Std. Dev.	Mean	Std. Dev.
1	Sphere (10 D)	7.98E-04	1.44E-04	4.10E-32	1.31E-32
2	Rosenbrock (10D)	8.88E+00	4.36E+01	9.55E-04	3.72E-05
3	Quartic (30 D)	1.18E-01	6.70E-02	4.18E-03	2.14E-02
4	Ackley (10D)	3.70E-01	6.35E-01	1.42E-14	2.25E-15
5	Schaffer (2D)	2.69E-04	2.60E-04	0.00E+00	0.00E+00
6	Easom (2D)	1.00E+00	2.74E-06	3.41E-04	2.29E-04
7	Rastrigin (10D)	1.93E+01	1.01E+01	0.00E+00	0.00E+00
8	Shekel (4D)	9.32E+00	3.44E-01	5.01E-04	2.26E-04
9	Langerman (10D)	7.06E-01	9.64E-05	3.21E-01	1.50E-01
10	Griewank (10D)	1.38E-01	8.28E-02	0.00E+00	0.00E+00
11	Branin (2D)	3.58E-07	0.00E+00	3.85E-04	2.95E-04
12	Sumpow (10D)	5.48E-04	2.73E-04	2.17E-04	3.05E-04
13	Bukin6 (2D)	7.35E-02	3.97E-02	1.02E-02	4.07E-03
14	Crossit (2D)	4.43E-02	4.49E-02	3.77E-04	2.57E-04
15	Drop (2D)	2.25E-01	1.14E-01	0.00E+00	0.00E+00
16	Shubert (2D)	7.88E+01	4.48E+01	4.93E-04	2.60E-04
17	Beale (2D)	1.58E-02	1.07E-01	4.21E-04	2.72E-04
18	McCorm (2D)	1.58E-01	1.32E-01	4.68E-04	2.60E-04
19	Camel6 (2D)	4.44E-01	2.85E-01	2.63E-04	2.94E-04
20	Boha1 (2D)	4.12E-04	2.65E-04	0.00E+00	0.00E+00
21	Colville (2D)	8.08E-04	1.76E-04	8.45E-04	1.45E-04
22	Powersum (2D)	8.31E-04	2.21E-04	6.42E-04	2.10E-04
23	Solomon (4D)	2.64E-03	1.39E-02	1.06E-15	3.07E-16
24	Alpine (2 D)	6.69E-04	2.35E-04	1.07E-16	3.93E-17
Total		2	4	22	20

According to the Mann–Whitney test results shown in Table 3.10, among the 24 benchmark functions, BAwSSR performed significantly better than the BBA in 19 functions, while no significant differences were observed for the remaining five functions. However, the values for BAwSSR and qABC reveal that the former was significantly better in 18 functions; while no significant differences noted for three functions, and qABC won the remaining three functions. Regarding SPSO2011, BAwSSR was significantly better in 21 functions; there were no statistically significant differences in two functions, and SPSO2011 performed significantly better in one function. Although this seems less than the accuracy values achieve, overall, BAwSSR performed significantly better, collectively, in 13 of the functions, with many of them having challenging features. Nonetheless, neither the proposed algorithm nor the other algorithms were able to converge in Langerman and Bukin6 functions.

As a conclusion, these findings indicate that the method used to enhance the standard BA in the BAwSSR, positively affected the performance over the algorithms used in this experiment.

Table 3.10 P-values using the Mann–Whitney test (a = 0.05) for accuracy acquired by BAwSSR over ABC, BBA, and SPSO2011

No.	Functions	BAwSSR–qABC		BAwSSR–SPSO2011		BAwSSR–BBA	
		p-value	Significant	p-value	Significant	p-value	Significant
1	Sphere (10D)	7.07E-18	Yes	7.07E-18	Yes	7.07E-18	Yes
2	Rosenbrock (10D)	7.07E-18	Yes	7.75E-09	Yes	7.07E-18	Yes
3	Quartic (30D)	1.08E-17	Yes	6.34E-17	Yes	7.07E-18	Yes
4	Ackley (10D)	1.86E-18	Yes	1.82E-18	Yes	1.86E-18	Yes
5	Schaffer (2D)	3.31E-20	Yes	3.31E-20	Yes	3.31E-20	Yes
6	Easom (2D)	7.40E-03	qABC	9.12E-20	Yes	2.40E-02	Yes
7	Rastrigin (10D)	3.31E-20	Yes	3.29E-20	Yes	3.31E-20	Yes
8	Shekel (4D)	1.43E-01	No	7.07E-18	Yes	2.92E-04	Yes
9	Langerman (10D)	4.22E-04	Yes	7.07E-18	Yes	6.97E-01	No
10	Griewank (10D)	3.31E-20	Yes	3.31E-20	Yes	3.31E-20	Yes
11	Branin (2D)	1.43E-02	qABC	3.31E-20	SPSO2011	8.87E-06	Yes

12	Sumpow (10D)	3.95E-05	Yes	1.58E-07	Yes	4.71E-10	Yes
13	Bukin6 (2D)	5.89E-08	Yes	5.85E-15	Yes	1.84E-07	Yes
14	Crossit (2D)	2.13E-03	Yes	7.07E-18	Yes	4.18E-01	No
15	Drop (2D)	3.31E-20	Yes	3.31E-20	Yes	3.31E-20	Yes
16	Shubert (2D)	2.99E-05	qABC	7.07E-18	Yes	8.11E-13	Yes
17	Beale (2D)	1.25E-04	Yes	6.03E-02	No	2.43E-01	No
18	McCorm (2D)	5.74E-01	No	7.07E-18	Yes	3.26E-01	No
19	Camel6 (2D)	6.72E-02	No	7.07E-18	Yes	9.14E-05	Yes
20	Boha1 (2D)	3.31E-20	Yes	3.31E-20	Yes	3.31E-20	Yes
21	Colville (2D)	1.35E-13	Yes	4.18E-01	No	3.90E-04	Yes
22	Powersum (2D)	2.03E-14	Yes	5.90E-07	Yes	2.63E-01	No
23	Solomon (4D)	6.06E-18	Yes	7.07E-18	Yes	7.07E-18	Yes
24	Alpine (2D)	7.07E-18	Yes	7.07E-18	Yes	7.07E-18	Yes
Total		BAwSSR: 18 qABC: 3 None: 3		BAwSSR: 21 SPSO2011: 1 None: 2		BAwSSR: 19 BBA: 0 None: 5	

3.5.2 SR and NFEs

The investigation of the SR result listed in the Table 3.11 achieved by the four algorithms involved in this study, displays that the proposed algorithm achieved the highest average SR of 92% with 100% in all but three functions. However, BAwSSR could not converge in the Bukin6 and Langerman functions obtaining 0% SR in both. However, qABC achieved poor result as well getting 2% while BBA got 30%

Table 3.11 SR of BAwSSR compared with BBA, SPSO2011, and qABC, based on NFEs obtained through 50 independent runs on test functions f1–f24

No.	Functions	qABC success rate	SPSO2011 success rate	BBA success rate	BAwSSR success rate
1	Sphere (10D)	100%	100%	100%	100%
2	Rosenbrock (10D)	0%	74%	0%	100%
3	Quartic (30 D)	36%	0%	0%	96%
4	Ackley (10D)	22%	74%	0%	100%
5	Schaffer (2D)	64%	100%	100%	100%
6	Easom (2D)	100%	100%	100%	100%
7	Rastrigin (10D)	98%	0%	0%	100%
8	Shekel (4D)	98%	100%	100%	100%
9	Langerman (10D)	2%	100%	30%	18%
10	Griewank (10D)	0%	0%	0%	100%
11	Branin (2D)	100%	0%	100%	100%
12	Sumpow (10D)	100%	100%	100%	100%
13	Bukin6 (2D)	4%	4%	0%	0%
14	Crossit (2D)	100%	100%	100%	100%

15	Drop (2D)	74%	100%	100%	100%
16	Shubert (2D)	100%	100%	14%	100%
17	Beale (2D)	76%	98%	100%	100%
18	McCorm (2D)	100%	100%	100%	100%
19	Camel6 (2D)	100%	100%	100%	100%
20	Boha1 (2D)	100%	100%	100%	100%
21	Colville (2D)	10%	100%	100%	100%
22	Powersum (2D)	16%	100%	100%	100%
23	Solomon (4D)	48%	96%	100%	100%
24	Alpine (2 D)	100%	100%	100%	100%
SR average		65%	77%	69%	92%

To draw the full picture of the proposed algorithm's performance, the NFEs need to be considered in this analysis. Briefly, the NFEs, or speed, represent the number of times the functions were executed by the algorithm while converging to the optimum; in other words, how fast was the algorithm able to converge to the minimum? Hence, lower NFE values mean that less time is needed to find the optimum. After considering the results shown in Table 3.12, which compares BAwSSR against the BBA, qABC, and SPSO2011, it can be clearly observed that the BAwSSR was faster than the BBA in 20 functions, whereas the latter was faster in only 3 functions, and there was no winner in 1 function. Additionally, among all the functions, BAwSSR converged faster than BBA in all the 50 runs of 14 functions. The comparison with qABC reveals similar findings, with BAwSSR excelling in 19 functions and with full performance in 10 functions for all 50 runs. However, qABC was faster in only five functions. Conversely, the figures for the comparison with SPSO2011 show that BAwSSR was faster in only 14 functions, while SPSO2011 was faster in 10 functions. However, in eight of the ten functions in which SPSO2011 excelled, the algorithm could neither find the optimum nor come close to it, which cannot be considered an indication of good performance. Yet, BAwSSR converged faster in all 50 runs in eight functions. Moreover, functions like Drop Sphere, Ackley, and Rastrigin needed only 102 NFEs, while functions like Griewank needed a little more, with 110 NFEs. The abovementioned figures demonstrate that BAwSSR performed better in terms of NFEs as well as the SR.

Table 3.12 Best performance of the BAwSSR, BBA, SPSO2011, and qABC for the NFEs obtained through 50 independent runs on test functions f1–f24

No.	Functions	Result of 50 runs		Result of 50 runs		Result of 50 runs	
		BAwSSR	qABC	BAwSSR	SPSO2011	BAwSSR	BBA
1	Sphere (10 D)	50	0	50	0	50	0
2	Rosenbrock (10D)	50	0	44	6	50	0
3	Quartic (30 D)	40	10	48	2	50	0
4	Ackley (10D)	50	0	50	0	50	0
5	Schaffer (2D)	50	0	50	0	50	0
6	Easom (2D)	19	31	0	50	45	4
7	Rastrigin (10D)	50	0	50	0	50	0
8	Shekel (4D)	7	43	0	50	25	25
9	Langerman (10D)	9	41	0	50	8	42
10	Griewank (10D)	50	0	50	0	50	0
11	Branin (2D)	30	20	50	0	49	1
12	Sumpow (10D)	49	1	50	0	50	0
13	Bukin6 (2D)	26	24	0	50	3	46
14	Crossit (2D)	39	11	2	48	37	4
15	Drop (2D)	50	0	0	50	50	0
16	Shubert (2D)	8	42	0	50	50	0
17	Beale (2D)	47	3	44	6	12	33
18	McCorm (2D)	46	4	0	50	36	12
19	Camel6 (2D)	17	33	0	50	24	19
20	Boha1 (2D)	50	0	50	0	50	0
21	Colville (2D)	46	4	7	43	50	0
22	Powersum (2D)	50	0	39	11	45	5
23	Solomon (4D)	50	0	50	0	50	0
24	Alpine (2 D)	50	0	50	0	50	0
Total		BAwSSR: 19 qABC: 5		BAwSSR: 14 SPSO2011:10		BAwSSR: 20 BBA: 3 None: 1	

Nevertheless, to discover other aspects of the performance, such as robustness and reliability, this assessment needs to compute the mean and standard deviation figures. Table 3.13 indicates that BAwSSR performed better than the BBA, having lower mean values in 21 functions, and lower standard deviations in 20 functions. The interpretation of these figures confirms the conclusion that the BAwSSR algorithm performed consistently faster in finding the minimum. Similarly, in the comparison with qABC (Table 3.14), BAwSSR acquired lower mean and standard deviation values in 20 and 21 functions, respectively, suggesting that BAwSSR consistently performed faster. On the Contrary, the figures for BAwSSR and SPSO2011 in Table 3.15 demonstrate that the former performed faster in only 14 functions only, whereas SPSO2011 performed faster in 10 functions. However, in seven functions (Easom, Shekel, Langerman, Crossit, Drop, Shubert, and McCorm), the mean was lower due to SPSO2011 converging prematurely after 100 NFEs, with an optimum value far from the standard one. This phenomenon was discussed in Chapter 2 as one of the weaknesses of PSO in general. As stated above, this cannot be considered an indication of better performance as the calculated optimum by SPSO2011 had low accuracy. On the other hand, BAwSSR itself could not converge and was trapped in local optima in the Langerman and Bukin6 functions.

The overall conclusion is that the achieved improvement could be attributed to the use of the search space reduction technique in BAwSSR. The use of this method in the initialisation and global stages via the five scenarios, helped the algorithm to conduct more focused sampling rather than routinely repeating the same search procedure in every iteration. Moreover, the samples taken via the search space reduction technique eventually guided the neighbourhood search to locate the optimum in more promising locations.

Table 3.13 Means and standard deviations of NFEs obtained through 50 independent runs on test functions f1–f24

No.	Functions	BBA		BAwSSR	
		Mean	Std. Dev.	Mean	Std. Dev.
1	Sphere (10D)	9180.34	2588.57	102.00	0.00
2	Rosenbrock (10D)	500042.22	5.54	55705.44	2036.26
3	Quartic (30)	500068.04	1.96	160903.42	151059.84
4	Ackley (10D)	500039.02	3.61	102.00	0.00
5	Schaffer (2D)	5066.22	2679.38	102.00	0.00
6	Easom (2D)	4060.24	1381.89	1972.66	814.79
7	Rastrigin (10D)	500068.86	3.86	102.00	0.00
8	Shekel (4D)	62742.04	32020.36	51442.88	11641.98
9	Langerman (10D)	385919.18	184710.55	431486.76	150993.14

10	Griewank (10D)	500045.52	3.98	110.00	0.00
11	Branin (2D)	4115.46	1758.67	857.66	388.14
12	Sumpow (10D)	951.20	179.34	75.68	47.38
13	Bukin6 (2D)	500012.54	5.56	500031.94	14.00
14	Crossit (2D)	400.84	227.00	164.24	69.92
15	Drop (2D)	27392.88	30397.32	102.00	0.00
16	Shubert (2D)	460845.84	106902.97	10468.80	3658.40
17	Beale (2D)	654.84	294.37	954.96	349.25
18	McCorm (2D)	1319.68	451.89	598.82	332.08
19	Camel6 (2D)	1025.76	692.52	874.40	300.75
20	Boha1 (2D)	93133.46	93331.19	102.00	0.00
21	Colville (2D)	90739.72	32563.57	31410.62	17993.65
22	Powersum (2D)	74731.52	65000.37	11361.32	8803.96
23	Solomon (2D)	38934.28	39767.45	102	0
24	Alpine (2D)	3546.60	1631.45	102	0
Total		3	4	21	20

Table 3.14 Means and standard deviations of NFEs for qABC obtained through 50 independent runs on test functions f1–f24

No.	Functions	qABC		BAwSSR	
		Mean	Std. Dev.	Mean	Std. Dev.
1	Sphere (10D)	33446.14	9643.90	102.00	0.00
2	Rosenbrock (10D)	500048.88	25.90	55705.44	2036.26
3	Quartic (30)	377869.86	180710.93	160903.42	151059.84
4	Ackley (10D)	468309.60	70444.77	102.00	0.00
5	Schaffer (2D)	233117.16	235650.01	102.00	0.00
6	Easom (2D)	2650.48	4249.69	1972.66	814.79
7	Rastrigin (10D)	180555.66	92942.62	102.00	0.00
8	Shekel (4D)	43292.62	96619.86	51442.88	11641.98
9	Langerman (10D)	491350.46	60942.64	431486.76	150993.14
10	Griewank (10D)	500049.58	27.24	110.00	0.00
11	Branin (2D)	1052.04	885.35	857.66	388.14
12	Sumpow (10D)	150.00	0.00	75.68	47.38
13	Bukin6 (2D)	490635.52	57616.16	500031.94	14.00
14	Crossit (2D)	274.00	115.86	164.24	69.92
15	Drop (2D)	236593.44	196388.31	102.00	0.00
16	Shubert (2D)	5662.18	9603.01	10468.80	3658.40
17	Beale (2D)	180170.24	213103.40	954.96	349.25
18	McCorm (2D)	13572.06	17897.62	598.82	332.08
19	Camel6 (2D)	790.00	369.86	874.40	300.75
20	Boha1 (2D)	1872.26	1145.41	102.00	0.00
21	Colville (2D)	453006.78	141528.21	31410.62	17993.65
22	Powersum (2D)	463048.28	94270.83	11361.32	8803.96
23	Solomon (2D)	273926.74	238028.51	102.00	0.00
24	Alpine (2D)	2344.00	8793.92	102.00	0.00
Total		4	3	20	21

Table 3.15 Means and standard deviations of NFEs obtained through 50 independent runs on test functions f1–f24

No.	Functions	SPSO2011		BAwSSR	
		Mean	Std. Dev.	Mean	Std. Dev.
1	Sphere (10D)	5884.00	437.89	102.00	0.00
2	Rosenbrock (10D)	209694.00	190711.63	55705.44	2036.26
3	Quartic (30 D)	500000.00	0.00	160903.42	151059.84
4	Ackley (10D)	139692.00	213573.00	102.00	0.00
5	Schaffer (2D)	2122.00	544.53	102.00	0.00
6	Easom (2D)	100.00	0.00	1972.66	814.79
7	Rastrigin (10D)	500000.00	0.00	102.00	0.00
8	Shekel (4D)	100.00	0.00	51442.88	11641.98
9	Langerman (10D)	100.00	0.00	431486.76	150993.14
10	Griewank (10D)	500000.00	0.00	110.00	0.00
11	Branin (2D)	500000.00	0.00	857.66	388.14
12	Sumpow (10D)	1502.00	327.10	75.68	47.38
13	Bukin6 (2D)	490634.00	65562.00	500031.94	14.00
14	Crossit (2D)	100.00	0.00	164.24	69.92
15	Drop (2D)	100.00	0.00	102.00	0.00
16	Shubert (2D)	100.00	0.00	10468.80	3658.40
17	Beale (2D)	11656.00	69765.16	954.96	349.25
18	McCorm (2D)	100.00	0.00	598.82	332.08
19	Camel6 (2D)	120.00	44.72	874.40	300.75
20	Boha1 (2D)	4992.00	742.12	102.00	0.00
21	Colville (2D)	14160.00	4506.00	31410.62	17993.65
22	Powersum (2D)	23094.00	23524.03	11361.32	8803.96
23	Solomon (2D)	26946.00	96573.06	102.00	0.00
24	Alpine (2D)	2912.00	1601.83	102.00	0.00
Total		10	10	14	11 None = 3

To determine the significance of the improvements exhibited by the proposed algorithm, the Mann–Whitney test was carried out on the NFE figures, and the findings are presented in Table 3.16. The findings reveal that, in the comparison between BAwSSR and the BBA, BAwSSR's performance was significantly better in 19 functions, the BBA's was better in two, and neither was superior in three. Similarly, the comparison with qABC reveals that BAwSSR was significantly better in 17 functions, while qABC was better in five, and neither was better in two. On the other hand, the BAwSSR–SPSO2011 comparison indicates that BAwSSR was significantly better in 14 functions and SPSO2011 was better in 10. As noted above, this result cannot be considered an indication of good performance. Nonetheless, the result presents convincing evidence for the effect of the technique used in the proposed algorithm.

Table 3.16 P-values using the Mann–Whitney test (a = 0.05) for NFEs acquired by BAwSSR over the BBA

No.	Functions	BAwSSR–qABC		BAwSSR–SPSO2011		BAwSSR–BBA	
		P-value	Significant	P-value	Significant	P-value	Significant
1	Sphere (10D)	3.31E-20	Yes	1.72E-20	Yes	3.28E-20	Yes
2	Rosenbrock (10D)	6.84E-18	Yes	3.05E-11	Yes	6.69E-18	Yes
3	Quartic (30 D)	4.58E-08	Yes	1.73E-17	Yes	6.21E-18	Yes
4	Ackley (10D)	3.30E-20	Yes	1.59E-20	Yes	3.11E-20	Yes
5	Schaffer (2D)	3.30E-20	Yes	1.72E-20	Yes	3.30E-20	Yes
6	Easom (2D)	1.21E-02	qABC	2.50E-20	SPSO2011	2.11E-13	Yes
7	Rastrigin (10D)	3.31E-20	Yes	1.59E-23	Yes	3.16E-20	Yes
8	Shekel (4D)	2.10E-10	qABC	2.55E-20	SPSO2011	2.98E-01	No
9	Langerman (10D)	2.65E-08	qABC	2.46E-20	SPSO2011	1.46E-09	BBA
10	Griewank (10D)	3.30E-20	Yes	1.59E-23	Yes	3.04E-20	Yes
11	Branin (2D)	9.96E-02	No	2.44E-20	Yes	3.41E-16	Yes
12	Sumpow (10D)	5.98E-21	Yes	1.24E-18	Yes	1.69E-18	Yes
13	Bukin6 (2D)	2.17E-01	No	3.63E-20	SPSO2011	3.88E-13	BBA
14	Crossit (2D)	1.42E-07	Yes	8.39E-18	SPSO2011	5.54E-09	Yes
15	Drop (2D)	3.31E-20	Yes	1.59E-23	SPSO2011	3.31E-20	Yes
16	Shubert (2D)	7.67E-10	qABC	2.38E-20	SPSO2011	6.34E-18	Yes
17	Beale (2D)	1.36E-14	Yes	3.55E-12	Yes	3.23E-05	BBA
18	McCorm (2D)	1.14E-12	Yes	2.32E-20	SPSO2011	4.92E-08	Yes
19	Camel6 (2D)	2.02E-02	qABC	2.86E-19	SPSO2011	1.71E-01	No
20	Boha1 (2D)	3.19E-20	Yes	1.74E-20	Yes	3.31E-20	Yes
21	Colville (2D)	7.63E-14	Yes	2.96E-12	SPSO2011	3.77E-15	Yes
22	Powersum (2D)	7.06E-18	Yes	1.07E-03	Yes	8.59E-12	Yes
23	Solomon (2D)	3.30E-20	Yes	1.78E-20	Yes	3.31E-20	Yes
24	Alpine (2D)	3.11E-20	Yes	1.76E-20	Yes	3.28E-20	Yes
Total		BAwSSR: 17 qABC: 5 None: 2		BAwSSR: 14 SPSO2011: 10 None: 0		BAwSSR: 19 BBA: 2 None: 3	

3.6 Summary

In this chapter, a new enhancement to the BA optimisation method was proposed. The core of this proposal is based on the gradual reduction of the search space, a technique borrowed from the numerical method of bracketing. This enhancement was applied to the initial and global search stages of the BA due to the researcher’s assumption that the neighbourhood search stage job is large extent dependent on these two stages. The proposed method was tested in 24 benchmark functions with a wide variety of surface topography and complexity. It was then compared with the latest versions of two of the highly popular algorithms, SPSO2011 and qABC. According to the Mann–Whitney statistical significance test, the results demonstrated significantly better performance by the proposed method in at least three-quarters of the functions tested in terms of accuracy as well as speed. The introduced method was able to find the exact minimum at a remarkably high speed, even for functions such as Ackley, Rastrigin, and Griewank, which create difficulties for many algorithms due to their highly pocketed topography. Moreover, it is worth noting that the test was carried out with a range of functions that are mostly differentiable and with limited dimensionality, where more than half of the functions were only two-dimensional. This implies the need for more thorough testing of the proposed method in a wider range of test functions.

Chapter 4 BEES ALGORITHM IMPROVEMENT USING DOMAIN SEGMENTATION (BADS)

4.1 Preliminaries

The BA, like many other metaheuristic algorithms, depends on randomisation to explore more solutions. This approach creates one of the problems hindering optimisation algorithms' search for the optimum—the inability of the extracted sample to be evenly distributed throughout the search space or being scattered nearby. This renders the generated samples biased, as it is highly likely that certain areas of the search space will be ignored. Hence, more processing time and computational resources will be needed, eventually causing the search process for the optima to become exceedingly costly. In general, more representative samples will be needed to achieve an exhaustive search, thus, smarter optimised search.

4.2 Domain Segmentation Sampling Method

To resolve the aforementioned problem, this chapter introduces a new technique based on segmenting the search space and sampling from each segment. The BA is a well-known algorithm and has been subject to a wide variety of developments and improvements by many researchers. To the best of this researcher's knowledge, hardly any BA improvements have been employed in the segmentation of the search space for the sake of diversifying the obtained solution. With the current sampling procedure of the BBA, there is no guarantee that the extracted sample will not ignore certain parts of the search space. With search space partitioning, each section will be independent from the others. The proposed algorithm will be applied to two of the BA stages: initialisation and global search. As mentioned in Chapter 3, these two stages have undergone fewer improvement attempts by researchers (Hussein *et al.*, 2014), so giving them more consideration is worthwhile. This focus is also due to the author's belief that these two stages have more impact in the BA's overall performance than does the neighbourhood search stage. In truth, as discussed in Chapter 3, neighbourhood search is highly dependent on these two stages. However, solely dividing the search space into independent segments might introduce its own problems if each different segment will be used to generate one different sample with all its parameters are taken from the same segment. The arising problem is related to the way the parameter values for some benchmarking or real-life engineering problems are distributed throughout the search space. Although, segmentation has proven to be effective in problems where parameters have the same value, but for other numerical benchmark functions and engineering design problems, the values of the parameters are not identical which means they might be located in different areas of the search space and hence different segments; thus, taking a full sample from one segment at a time means it is likely to miss some parameter values ending up with the search trapped at some point and cannot move forward. The suggested solution is to provide the option to extract the sample in two approaches:

1. The whole sample is extracted from one independent segment; the number of segments is equal to the number of samples (Figure 4.1). For example, for the function Griewank, where search space is [-600, 600], the segment length will be calculated as follows:

   $$\text{Segment length } (S) = L/n = 1200/10 = 120 \qquad (4.1)$$

   where n is the number of samples = 10, and L is the length space, calculated as

   $$L = (600 - (-600)) = 1200 \qquad (4.2)$$

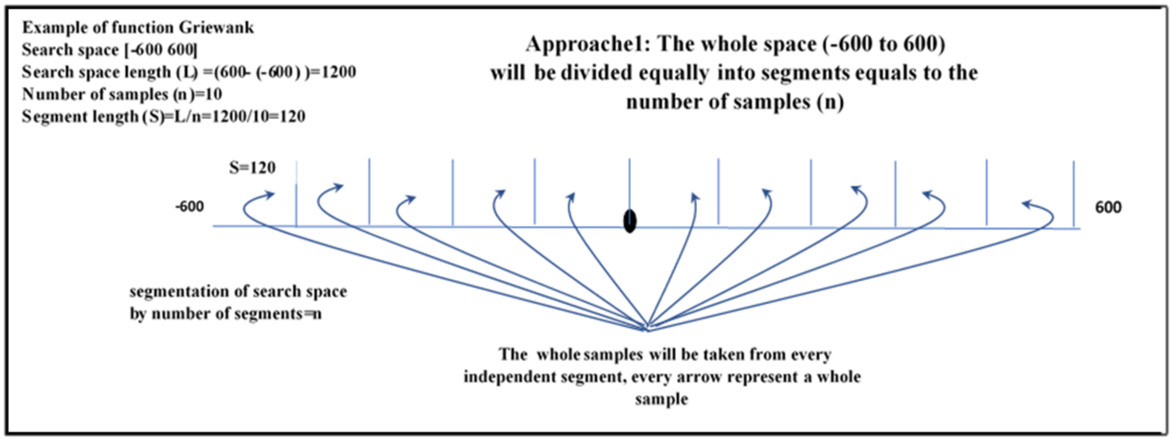

Example of function Griewank
Search space [-600 600]
Search space length (L) =(600- (-600))=1200
Number of samples (n)=10
Segment length (S)=L/n=1200/10=120
Approache1: The whole space (-600 to 600) will be divided equally into segments equals to the number of samples (n)
S=120
-600
600
segmentation of search space by number of segments=n
The whole samples will be taken from every independent segment, every arrow represent a whole sample

Figure 4.1 First sampling approach from segmented search space

2. Every parameter of the same sample will be taken from a different segment; the number of segments will be equal to the number of variables (Figure 4.2). For example, for function Colville, the search space is [-10, 10], and the number of parameters (D) = 4:

   Search space length $L = (10 - (-10)) = 20$ (4.3)

   Length of Segment $S = L/D = 20/4 = 5$ (4.4)

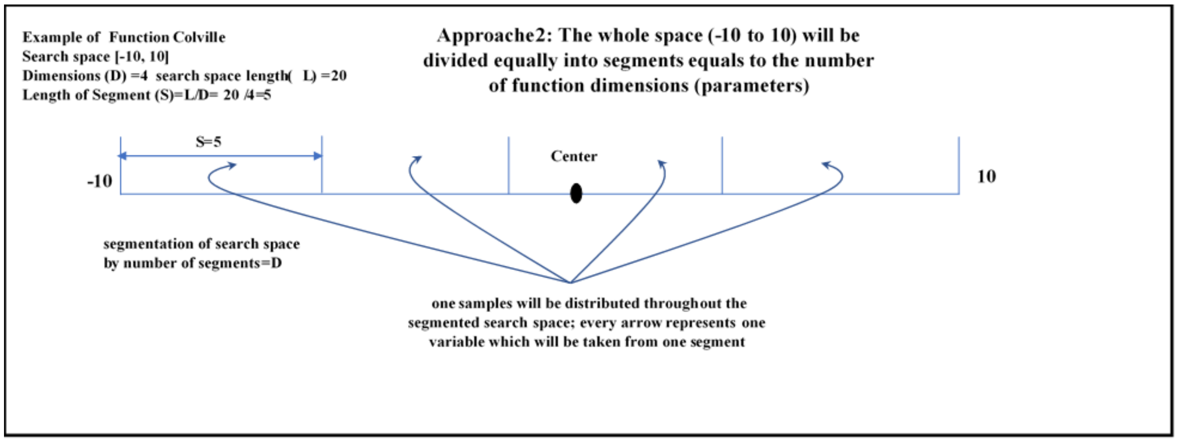

Example of Function Colville
Search space [-10, 10]
Dimensions (D) =4 search space length(L) =20
Length of Segment (S)=L/D= 20 /4=5
Approache2: The whole space (-10 to 10) will be divided equally into segments equals to the number of function dimensions (parameters)
S=5
Center
-10
10
segmentation of search space by number of segments=D
one samples will be distributed throughout the segmented search space; every arrow represents one variable which will be taken from one segment

Figure 4.2 Second approach sampling from segmented search space

However, to optimise the above two search approaches to focus on promising areas of search space, these areas will be tracked according to the best parameters found that give the closest optimum to the standard. When these promising areas are identified, the search for the optimum will be reduced to these areas. This is an ongoing process that will be performed repeatedly to optimise the search. Thus, the proposed method will be named the BA domain (BADS) segmentation.

4.3 Search Space Mobile Subset Sampling

Another sampling method used in this proposed enhancement in this chapter is the mobile small subset of the search space. In this method, a small subset of the search will be used to extract the sample. The search will be limited to small subset, generated from the whole space to concentrate the effort for generating sample. at first, the subset will start from the left end of the domain up to the added small subset. For example, for function Colville, where search space is limited between [-10, 10] (Figure 4.3):

at the start of the search $a_i = -10,\ i = 1,2, \dots n$ (4.5)

where a_I is the first segment of the search space , n is the number of samples.

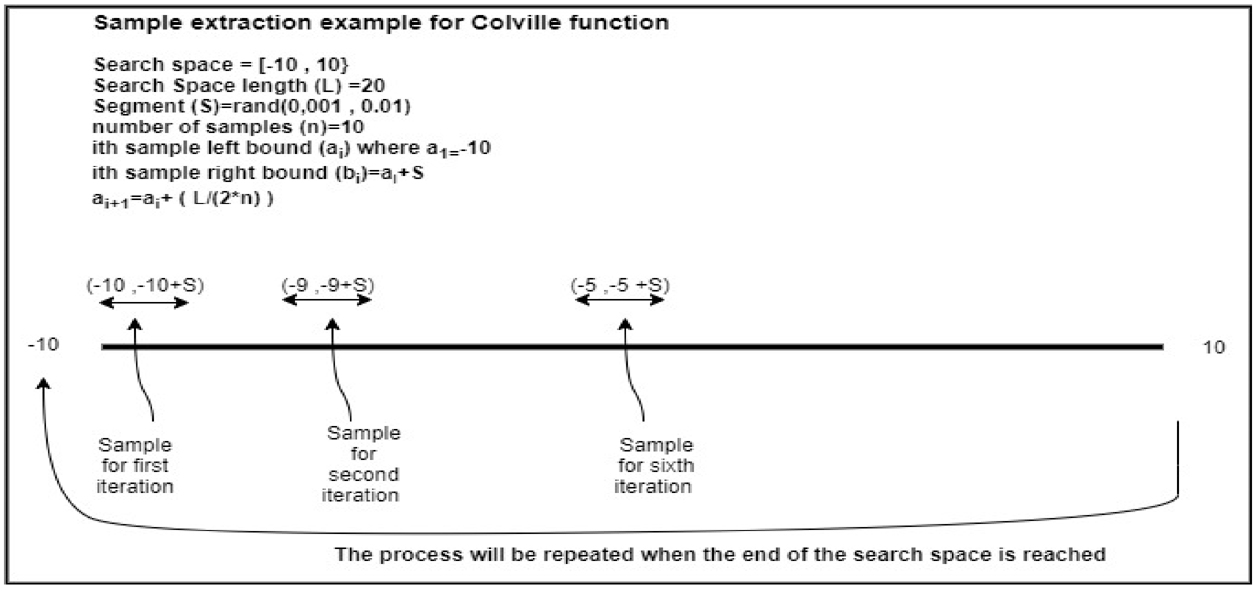
the length of the search space L = (10− (−10)) = 20 (4.6)

$b_i = a_i + S,\ \ where\ S =$ randum (0.001, 0.01) (4.7)

The next subset will be calculated as:

$a_{i+1} = a_i + L/(2 * n)$, L is (4.8)

The process will continue until the right end of the search space is reached; the search then will return from the left end where it started.


Sample extraction example for Colville function
Search space = [-10 , 10}
Search Space length (L) =20
Segment (S)=rand(0,001 , 0.01)
number of samples (n)=10
ith sample left bound (a_i) where a_1=-10
ith sample right bound (b_i)=a_i+S
a_{i+1}=a_i+ (L/(2*n))
(-10 ,-10+S)
(-9 ,-9+S)
(-5 ,-5 +S)
-10
10
Sample for first iteration
Sample for second iteration
Sample for sixth iteration
The process will be repeated when the end of the search space is reached
Figure 4.3 Sampling from a mobile subset of search space

4.4 Experiment Setup

An improved version of the BA based on the proposed method described in Sections 4.2 and 4.3 was tested, and the results are reported here. The test was carried out on the same list of benchmark functions used in Chapter 3, but with different parameter settings for the proposed algorithm. These settings are described in Table 4.1. However, the parameter settings for the BBA, qABC, and SPSO2011 remained the same. Furthermore, the exact performance metrics, which consist of the accuracy, SR, and NFE, were used. These metrics were applied to the results obtained from 50 runs, with the same stopping criteria and evaluation procedure as described in Chapter 3. Similarly, the Mann–Whitney statistical significance test was performed on the acquired test results and was used to evaluate and compare the performance of all the algorithms involved.

Table 4.1 List of parameter values used for testing BADS

No.	Functions	n	m	nsp	e	nep	ngh	stlim
1	Sphere (10D)	13	4	10	2	30	0.03	10
2	Rosenbrock (10D)	100	4	10	2	30	0.0001	10
3	Quartic (30D)	13	4	10	2	30	0.005	10
4	Ackley (10D)	13	4	10	2	30	0.00001	10

5	Schaffer (2D)	13	4	10	2	30	0.01	10
6	Easom (2D)	100	4	10	2	30	0.00009	10
7	Rastrigin (10D)	13	4	10	2	30	0.003	10
8	Shekel (4D)	1,000	4	10	2	30	0.001	10
9	Langerman (10D)	100	4	10	2	30	0.09	10
10	Griewank (10D)	17	4	10	2	30	0.001	10
11	Branin (2D)	15	4	10	2	30	0.01	10
12	Sumpow (10D)	20	4	10	2	30	0.000`	10
13	Bukin6 (2D)	11	4	10	2	30	0.01	10
14	Crossit (2D)	15	4	10	2	30	1	10
15	Drop (2D)	13	4	10	2	30	0.01	10
16	Shubert (2D)	20	4	10	2	30	0.0005	10
17	Beale (2D)	1,000	4	10	2	30	0.45	10
18	McCorm (2D)	100	4	10	2	30	0.05	10
19	Camel6 (2D)	13	4	10	2	30	0.005	10
20	Boha1 (2D)	13	4	10	2	30	0.05	10
21	Colville (2D)	13	4	10	2	30	0.0005	10
22	Powersum (2D)	13	4	10	2	30	0.01	10
23	Solomon (2D)	13	4	10	2	30	0.001	10
24	Alpine (2D)	100	4	10	2	30	0.0001	10

4.5 Results and Discussion

4.5.1 Solution Quality (Accuracy)

Table 4.2 presents the overall difference in performance for every function in 50 runs by the involved algorithms. As indicated by these values, the proposed algorithm, BADS, exhibited more instances of improved performance than all other algorithms in no less than two-thirds of the functions tested. The best result was in testing BADS against SPSO2011, where BADS achieved better accuracy performance in 22 of the functions (91%), while SPSO2011 performed better in only 1 function. The result of testing against the BBA was next in terms of best accuracy performance, as BADS performed better in 21 functions (90%), while the BBA was better in only 3 functions. However, qABC showed slightly better performance than SPSO2011 and obtained better results in 5 functions, while BADS was superior in 19 functions (80%). In general, the results demonstrate outstanding performance by the proposed algorithm in, for example, Rosenbrock (10D), Quartic (30D), Ackley (10D), Rastrigin (10D), Griewank (10D), and Solomon (2D) functions, where it performed better in most of the 50 runs. These functions used to represent a challenge for many optimisation algorithms due to their complex topography.

Table 4.2 Best performance figures for BADS, BBA, qABC, and SPSO2011 for accuracy values

No.	Functions	Result of 50 runs		Result of 50 runs		Result of 50 runs	
		BADS	qABC	BADS	SPSO2011	BADS	BBA
1	Sphere (10D)	42	8	41	9	46	4
2	Rosenbrock (10D)	50	0	50	0	50	0
3	Quartic (30D)	49	1	48	2	50	0
4	Ackley (10D)	49	1	50	0	50	0
5	Schaffer (2D)	45	5	44	6	43	7
6	Easom (2D)	27	23	50	0	35	15
7	Rastrigin (10D)	49	1	50	0	50	0
8	Shekel (4D)	34	16	50	0	47	3
9	Langerman (10D)	5	45	49	1	0	50
10	Griewank (10D)	50	0	50	0	50	0
11	Branin (2D)	14	36	0	50	37	13
12	Sumpow (10D)	39	11	42	8	45	5
13	Bukin6 (2D)	1	49	25	25	1	49
14	Crossit (2D)	36	14	50	0	31	19
15	Drop (2D)	45	5	50	0	42	8
16	Shubert (2D)	16	34	50	0	46	4
17	Beale (2D)	37	13	27	23	31	19
18	McCorm (2D)	29	21	50	0	27	23
19	Camel6 (2D)	21	29	50	0	28	22
20	Boha1 (2D)	38	12	35	15	43	7
21	Colville (2D)	48	2	46	4	49	1
22	Powersum (2D)	46	4	31	19	18	32
23	Solomon (2D)	50	0	46	4	50	0
24	Alpine (2D)	46	4	47	3	48	2
Total		BADS: 19		BADS: 22		BADS: 21	
		qABC: 5		SPSO2011: 1		BBA: 3	
				None: 1			

Figures 4.4–4.12 are the performance charts in terms of accuracy by all the algorithms involved.

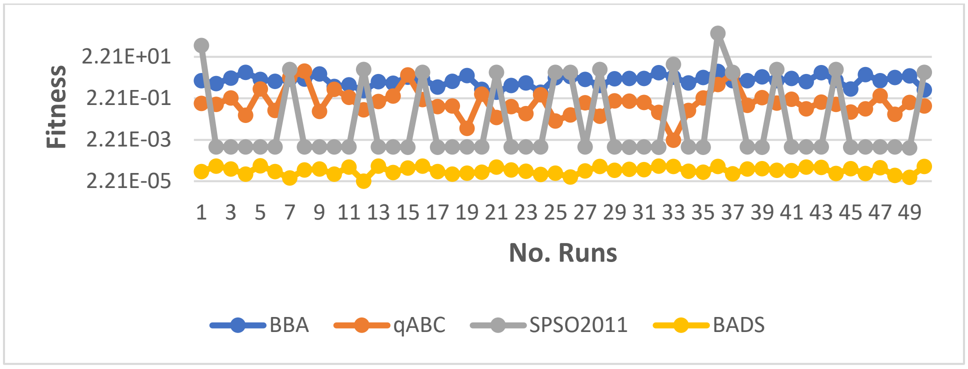


Figure 4.4 Result of 50 runs for the BBA, BADS, qABC, and SPSO2011 on Rosenbrock 10D function

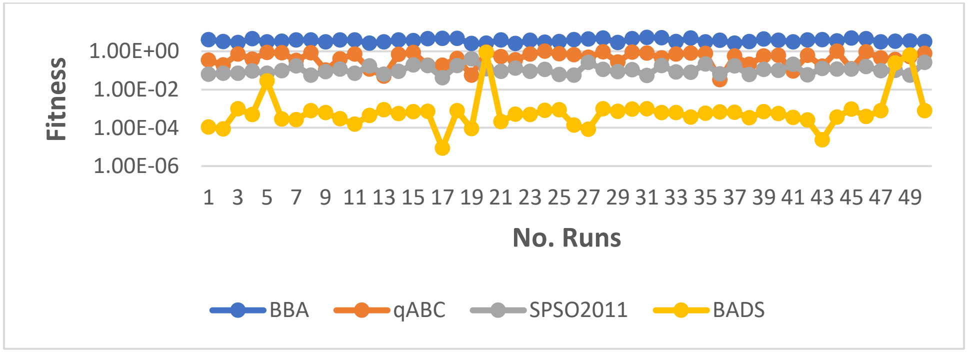


Figure 4.5 Result of 50 runs for the BBA, BADS, qABC, and SPSO2011 on Quatric 30D function

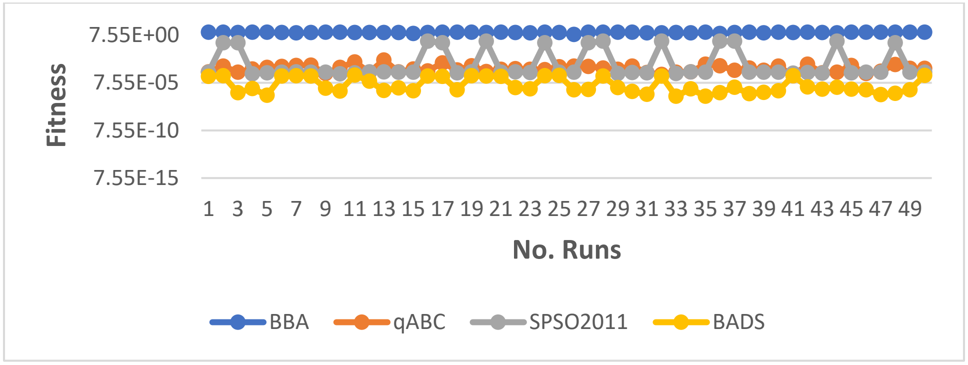


Figure 4.6 Result of 50 runs for the BBA, BADS, qABC, and SPSO2011 on Ackley 10D function

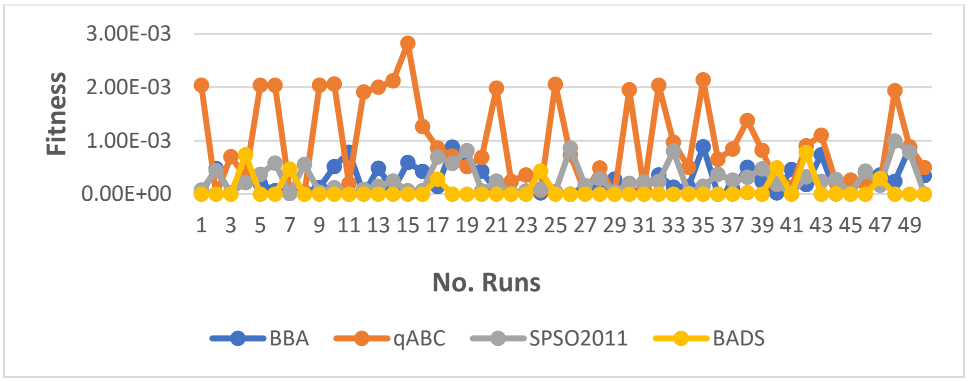


Figure 4.7 Result of 50 runs for the BBA, BADS, qABC, and SPSO2011 on Schaffer 2D function

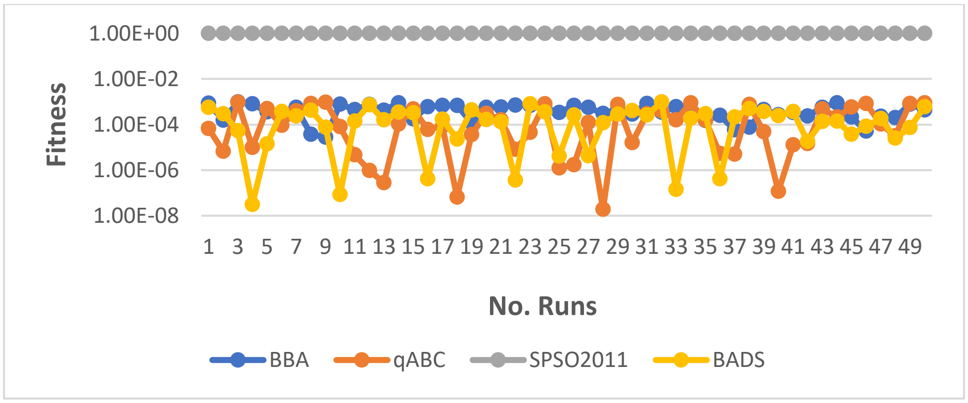


Figure 4.8 Result of 50 runs for the BBA, BADS, qABC, and SPSO2011 on Easom 2D function

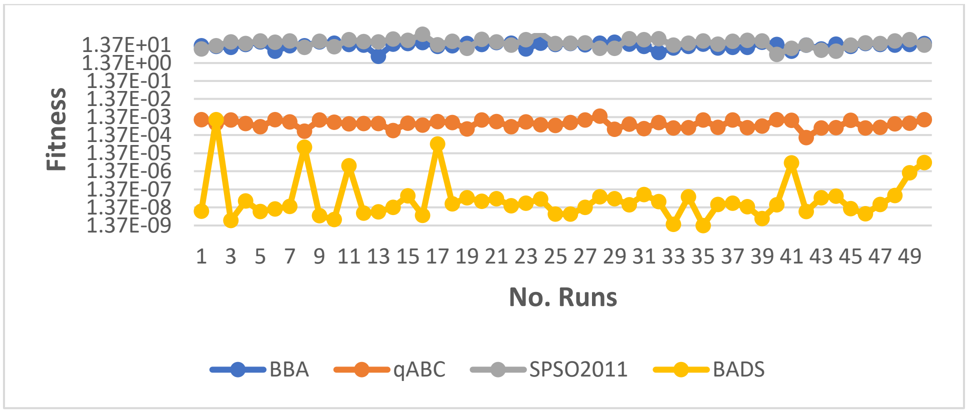


Figure 4.9 Result of 50 runs for the BBA, BADS, qABC, and SPSO2011 on Rastrigin 2D function

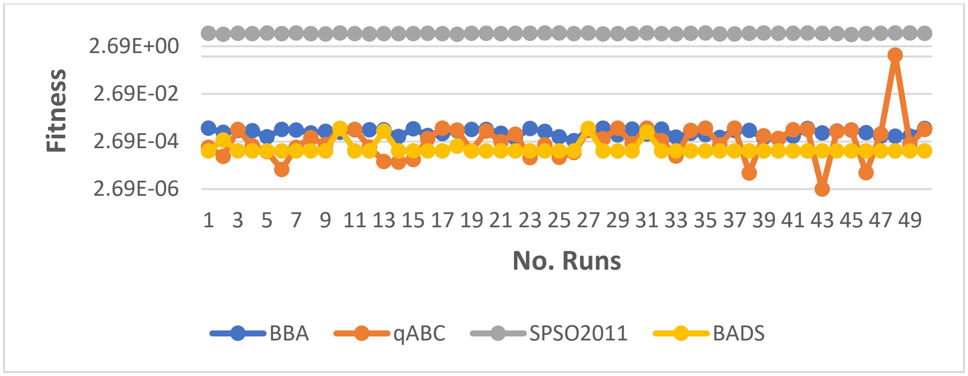


Figure 4.10 Result of 50 runs for the BBA, BADS, qABC, and SPSO2011 on Shekel 4D function

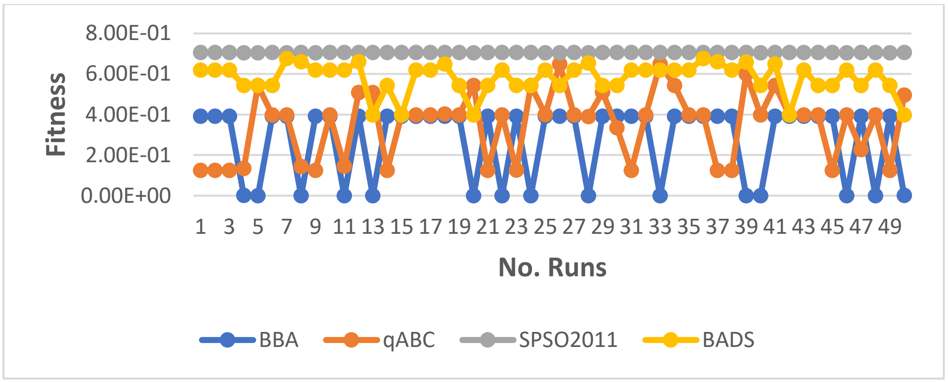


Figure 4.11 Result of 50 runs for the BBA, BADS, qABC, and SPSO2011 on Langerman 10D function

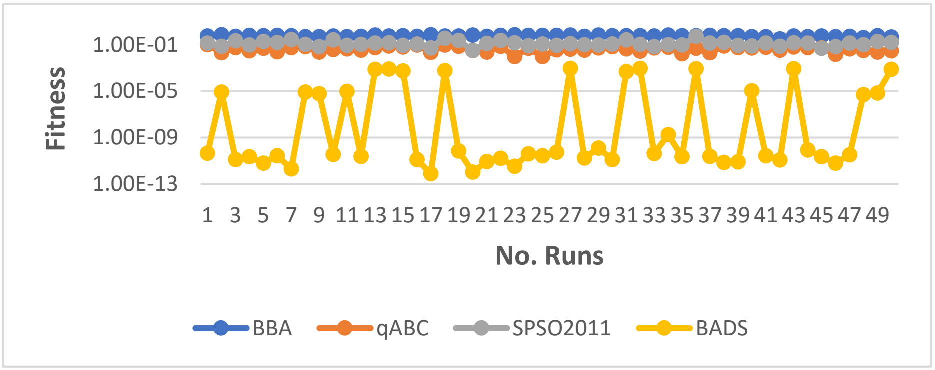


Figure 4.12 Result of 50 runs for the BBA, BADS, qABC, and SPSO2011 on Griewank 10D function

Nonetheless, these findings must be validated using the means and standard deviations obtained for the performance figures to inspect the consistency and stability of the suggested technique. However, as mentioned in Chapter 3, in certain cases, the standard deviation values might not be true reflections of consistent behaviour, as one algorithm might consistently produce same optimum value even though its worse than its counterparts in terms of accuracy.

The values in Table 4.3 compare the means and standard deviations of the BBA and BADS. It can be noticed that, while the mean values for BADS were better than BBA in 21 functions, the standard deviations did not perform equally well, having better values in only 16 functions. However, the values for two functions were eliminated because of their deficient performance. In the Langerman and Bukin6 functions, BADS was stuck in local optima and could not converge, where as in Powersum function BBA performed than BADS. The overall result suggests that BADS performed consistently better in 13 functions, where the means and standard deviations were collectively better than those for BBA. This comprises slightly more than half of the functions tested, which suggests that the proposed algorithm was not fully consistent.

Table 4.3 Means and standard deviations of best accuracy values for BADS and the BBA through 50 independent runs on functions f1–f24

No.	Functions	BBA		BADS	
		Mean	Std. Dev.	Mean	Std. Dev.
1	Sphere (10D)	8.19E-04	1.27E-04	4.19E-04	2.93E-04
2	Rosenbrock (10D)	1.80E+00	9.23E-01	3.53E-04	3.20E-05
3	Quartic (30D)	3.70E+00	7.34E-01	2.45E-02	1.29E-01
4	Ackley (10D)	1.38E+01	1.25E+00	1.95E-04	1.96E-04
5	Schaffer (2D)	2.57E-04	2.61E-04	3.53E-04	2.19E-04
6	Easom (2D)	4.56E-04	2.64E-04	3.53E-04	2.28E-04
7	Rastrigin (10D)	1.33E+01	3.84E+00	9.14E-05	2.56E-04
8	Shekel (4D)	6.67E-04	1.96E-04	3.53E-04	1.55E-04
9	Langerman (10D)	2.74E-01	1.79E-01	7.00E-01	1.37E-02
10	Griewank (10D)	5.68E-01	7.78E-02	3.36E-04	3.57E-04
11	Branin (2D)	6.43E-04	1.96E-04	3.99E-04	3.10E-04
12	Sumpow (10D)	6.86E-04	2.44E-04	3.53E-04	3.17E-04
13	Bukin6 (2D)	1.59E-02	4.93E-03	5.22E-02	2.68E-02
14	Crossit (2D)	4.23E-04	2.71E-04	3.53E-04	2.98E-04
15	Drop (2D)	4.06E-04	2.48E-04	3.53E-04	2.86E-04
16	Shubert (2D)	5.24E-03	4.89E-03	5.49E-04	2.82E-04
17	Beale (2D)	4.82E-04	2.74E-04	3.53E-04	3.09E-04
18	McCorm (2D)	5.26E-04	2.92E-04	3.53E-04	2.78E-04
19	Camel6 (2D)	4.87E-04	2.86E-04	3.53E-04	2.41E-04
20	Boha1 (2D)	4.68E-04	2.92E-04	3.53E-04	2.30E-04
21	Colville (2D)	9.28E-04	7.65E-05	3.53E-04	2.68E-04
22	Powersum (4D)	6.84E-04	2.26E-04	7.61E-04	2.08E-04
23	Solomon (2D)	5.49E-04	2.57E-04	3.53E-04	2.15E-04
24	Alpine (2D)	6.25E-04	2.41E-04	1.33E-04	1.23E-04

Total	3	8	21	16

The values in Table 4.4, which compares the means and standard deviations of qABC and BADS, indicate clearly that BADS exhibited more stable performance than qABC. BADS achieved better means and standard deviations in 19 functions. The standard deviation values for the Langerman and Bukin6 functions were excluded even though BADS obtained better standard deviations, as qABC acquired better solutions in most of the runs (45). However, neither algorithm could converge within the limit of NFEs.

Table 4.4 Means and standard deviations of accuracy values through 50 runs on functions f1–f24 for BADS and qABC

No.	Functions	qABC		BADS	
		Mean	Std. Dev.	Mean	Std. Dev.
1	Sphere (10D)	6.24E-04	2.49E-04	2.43E-04	2.96E-04
2	Rosenbrock (10D)	4.09E-01	8.85E-01	7.65E-05	2.72E-05
3	Quartic (30D)	5.23E-01	2.88E-01	3.57E-02	1.50E-01
4	Ackley (10D)	3.11E-03	3.05E-03	1.44E-04	1.80E-04
5	Schaffer (2D)	9.65E-04	8.04E-04	7.00E-05	1.85E-04
6	Easom (2D)	2.73E-04	3.28E-04	2.33E-04	2.23E-04
7	Rastrigin (10D)	6.22E-04	2.78E-04	2.15E-05	1.38E-04
8	Shekel (4D)	2.34E-02	1.61E-01	1.72E-04	1.99E-04
9	Langerman (10D)	3.50E-01	1.63E-01	5.80E-01	7.44E-02
10	Griewank (10D)	4.65E-02	2.19E-02	1.49E-04	3.02E-04
11	Branin (2D)	2.63E-04	2.76E-04	4.20E-04	2.82E-04
12	Sumpow (10D)	3.82E-04	1.12E-04	1.62E-04	2.75E-04
13	Bukin6 (2D)	2.81E-02	1.61E-02	8.74E-02	2.45E-02
14	Crossit (2D)	5.65E-04	3.01E-04	3.42E-04	2.67E-04
15	Drop (2D)	1.12E-02	2.17E-02	1.20E-04	2.71E-04
16	Shubert (2D)	2.69E-04	3.13E-04	4.37E-04	2.50E-04
17	Beale (2D)	1.98E-03	3.49E-03	4.22E-04	2.81E-04
18	McCorm (2D)	5.06E-04	2.83E-04	4.43E-04	2.37E-04
19	Camel6 (2D)	4.01E-04	3.26E-04	4.26E-04	2.70E-04
20	Boha1 (2D)	3.20E-04	2.78E-04	1.32E-04	2.21E-04
21	Colville (2D)	3.43E-02	3.10E-02	3.88E-04	3.54E-04
22	Powersum (4D)	7.13E-03	8.58E-03	7.61E-04	2.08E-04
23	Solomon (2D)	4.80E-02	4.88E-02	2.96E-05	1.16E-04
24	Alpine (2D)	5.58E-04	2.84E-04	1.14E-04	9.20E-05
Total		5	4	19	20

The data in Table 4.5, which compares the performance of SPSO2011 against BADS, indicate similar discrepancies between these two statistical measures in Table 4.3. The BADS mean figures were better in 22 functions, whereas its standard deviations were better in only 17 functions. However, the figures for Bukin6 were not considered because it performed poorly and did not converge to the optimum at all, winning only half of the 50 runs against SPSO2011. Similarly, Langerman should be excluded as BADS could not converge within the NFEs limit. Moreover, many functions performed well in most of the 50 runs, such as Sphere, Quartic, and Easom, but the standard deviations were not better than those of SPSO2011, which suggests performance inconsistency. In general, BADS did not show stable behaviour in almost one-third of the functions. Additionally, ngh parameter tuning was needed to improve the result, demonstrating sensitive behaviour by BADS.

Table 4.5 Means, and standard deviations of accuracy values obtained through 50 runs on test functions f1–f24 for BADS and SPSO2011

No.	Functions	SPSO2011		BADS	
		Mean	Std. Dev.	Mean	Std. Dev.
1	Sphere (10D)	7.98E-04	1.44E-04	2.43E-04	2.96E-04
2	Rosenbrock (10D)	8.88E+00	4.36E+01	7.65E-05	2.72E-05
3	Quartic (30D)	1.18E-01	6.70E-02	3.57E-02	1.50E-01
4	Ackley (10D)	3.70E-01	6.35E-01	1.44E-04	1.80E-04
5	Schaffer (2D)	2.69E-04	2.60E-04	7.00E-05	1.85E-04
6	Easom (2D)	1.00E+00	2.74E-06	2.33E-04	2.23E-04
7	Rastrigin (10D)	1.93E+01	1.01E+01	2.15E-05	1.38E-04
8	Shekel (4D)	9.32E+00	3.44E-01	1.72E-04	1.99E-04
9	Langerman (10D)	7.06E-01	9.64E-05	5.80E-01	7.44E-02
10	Griewank (10D)	1.38E-01	8.28E-02	1.49E-04	3.02E-04
11	Branin (2D)	3.58E-07	0.00E+00	4.20E-04	2.82E-04
12	Sumpow (10D)	5.48E-04	2.73E-04	1.62E-04	2.75E-04
13	Bukin6 (2D)	7.35E-02	3.97E-02	8.74E-02	2.45E-02
14	Crossit (2D)	4.43E-02	4.49E-02	3.42E-04	2.67E-04
15	Drop (2D)	2.25E-01	1.14E-01	1.20E-04	2.71E-04
16	Shubert (2D)	7.88E+01	4.48E+01	4.37E-04	2.50E-04
17	Beale (2D)	1.58E-02	1.07E-01	4.22E-04	2.81E-04
18	McCorm (2D)	1.58E-01	1.32E-01	4.43E-04	2.37E-04
19	Camel6 (2D)	4.44E-01	2.85E-01	4.26E-04	2.70E-04
20	Boha1 (2D)	4.12E-04	2.65E-04	1.32E-04	2.21E-04
21	Colville (2D)	8.08E-04	1.76E-04	3.88E-04	3.54E-04
22	Powersum (2D)	8.31E-04	2.21E-04	7.61E-04	2.08E-04
23	Solomon (4D)	2.64E-03	1.39E-02	2.96E-05	1.16E-04
24	Alpine (2 D)	6.69E-04	2.35E-04	1.14E-04	9.20E-05
Total		2	7	22	17

Although the above analysis and discussion suggest a proper level of improvement was achieved by the proposed algorithm, the Mann–Whitney test must be used to verify whether this improvement is statistically significant. The p-values in Table 4.6 reveal that BADS was significantly better than qABC in 17 functions, which is equivalent to the figures against the BBA. On the other hand, BADS was significantly better than SPSO2011 in 21 functions. The overall significant test results indicate that BADS exhibited significantly better performance in at least 70% of the functions tested. In general, BADS exhibited the ability to calculate accurate optimum with a level of consistency in the average. Nonetheless, considering the number of functions used in the experiment, it is highly unlikely that the improved performance of the BA happened arbitrarily and not as an effect of employing the domain segmentation method.

Table 4.6 P-values using Mann–Whitney test (a = 0.05) for accuracy acquired by BADS over qABC, BBA, and SPSO2011

No.	Functions	BADS–qABC		BADS–SPSO2011		BADS–BBA	
		P-value	Significant	P-value	Significant	P-value	Significant
1	Sphere (10D)	2.60E-08	Yes	4.76E-14	Yes	2.26E-14	Yes
2	Rosenbrock (10D)	7.07E-18	Yes	7.03E-18	Yes	7.07E-18	Yes
3	Quartic (30D)	9.92E-16	Yes	2.95E-14	Yes	7.07E-18	Yes
4	Ackley (10D)	1.01E-17	Yes	6.92E-18	Yes	7.07E-18	Yes
5	Schaffer (2D)	1.82E-14	Yes	3.23E-11	Yes	3.55E-11	Yes
6	Easom (2D)	6.92E-01	No	9.12E-20	Yes	2.01E-05	Yes
7	Rastrigin (10D)	1.20E-16	Yes	7.02E-18	Yes	7.07E-18	Yes
8	Shekel (4D)	2.45E-03	Yes	7.07E-18	Yes	2.80E-14	Yes
9	Langerman (10D)	2.16E-13	qABC	7.07E-18	Yes	7.07E-18	BBA
10	Griewank (10D)	7.07E-18	Yes	7.07E-18	Yes	7.07E-18	Yes
11	Branin (2D)	3.14E-03	qABC	3.31E-20	SPSO2011	5.93E-05	Yes
12	Sumpow (10D)	2.25E-07	Yes	5.14E-10	Yes	5.05E-12	Yes
13	Bukin6 (2D)	2.05E-15	qABC	1.51E-01	No	7.12E-17	BBA
14	Crossit (2D)	4.56E-04	Yes	7.07E-18	Yes	1.03E-01	No
15	Drop (2D)	6.13E-12	Yes	7.07E-18	Yes	2.89E-10	Yes
16	Shubert (2D)	5.19E-04	qABC	7.07E-18	Yes	3.43E-13	Yes
17	Beale (2D)	1.52E-04	Yes	8.30E-02	No	2.93E-01	No
18	McCorm (2D)	2.37E-01	No	7.07E-18	Yes	3.06E-01	No
19	Camel6 (2D)	4.84E-01	No	7.07E-18	Yes	1.76E-01	No
20	Boha1 (2D)	1.42E-06	Yes	7.45E-09	Yes	1.34E-09	Yes
21	Colville (2D)	2.13E-16	Yes	1.03E-08	Yes	5.57E-14	Yes
22	Powersum (2D)	6.00E-13	Yes	2.91E-02	Yes	8.30E-02	No
23	Solomon (4D)	1.24E-16	Yes	7.12E-17	Yes	2.14E-16	Yes
24	Alpine (2D)	3.10E-12	Yes	1.07E-16	Yes	3.79E-16	Yes
Total		BADS: 17 qABC: 4 None: 3		BADS: 21 SPSO2011: 1 None: 2		BADS: 17 BBA: 2 None: 5	

4.5.2 SR and NFEs

When examining Table 4.7, it is evident that BADS achieved the highest SR average, with 91% obtained for all functions. This result was affected by the poor performance of Langerman and Bukin6, the two functions that could not converge getting 0% SR, suggesting that they were trapped in some local optima. Those two functions have always been too difficult to solve for all the proposed algorithms in this research. Additionally, the function Quartic achieved only 92% SR. The figures for the other algorithms are as follows: SPSO2011 77%, BBA 69%, and qABC 65%.

Table 4.7 SR of the BADS, BBA, SPSO2011, and qABC algorithms based on NFEs obtained through 50 runs on functions f1–f24

No.	Functions	qABC success rate	SPSO2011 success rate	BBA success rate	BADS success rate
1	Sphere (10 D)	100%	100%	100%	100%
2	Rosenbrock (10D)	0%	74%	0%	100%
3	Quartic (30D)	36%	0%	0%	92%
4	Ackley (10D)	22%	74%	0%	100%
5	Schaffer (2D)	64%	100%	100%	100%
6	Easom (2D)	100%	100%	100%	100%
7	Rastrigin (10D)	98%	0%	0%	100%
8	Shekel (4D)	98%	100%	100%	100%
9	Langerman (10D)	2%	100%	30%	0%
10	Griewank (10D)	0%	0%	0%	100%
11	Branin (2D)	100%	0%	100%	100%
12	Sumpow (10D)	100%	100%	100%	100%
13	Bukin6 (2D)	4%	4%	0%	0%
14	Crossit (2D)	100%	100%	100%	100%
15	Drop (2D)	74%	100%	100%	100%
16	Shubert (2D)	100%	100%	14%	100%
17	Beale (2D)	76%	98%	100%	100%
18	McCorm (2D)	100%	100%	100%	100%
19	Camel6 (2D)	100%	100%	100%	100%
20	Boha1 (2D)	100%	100%	100%	100%
21	Colville (2D)	10%	100%	100%	100%
22	Powersum (2D)	16%	100%	100%	100%
23	Solomon (4D)	48%	96%	100%	100%
24	Alpine (2 D)	100%	100%	100%	100%
SR average		65%	77%	69%	91%

As mentioned in Chapter 3, NFE measures how quickly an algorithm finds the optimum. It is a common indicator used in the literature on optimisation algorithms because it is more objective than CPU time. From Table 4.8, BADS outperformed the BBA and qABC, performing faster than the BBA in 21 functions and faster than qABC in 20 functions. While SPSO2011 appears to function efficiently, were it achieved better NFE values in 11 functions (or one-third of the total); nonetheless, its performance in eight of the functions was not an improvement, as the marginal difference for the optimum found by SPSO2011 was at least twice that of the standard optimum due to the premature convergence phenomenon that was highlighted in previous chapters. Yet, the overall inference from the figures is that BADS exhibited a noticeable improvement, outperforming the other algorithms involved in the testing. However, to evaluate the consistent and stability behaviour of the proposed algorithm, the means and standard deviations should be calculated to provide adequate performance comparisons with the other algorithms.

Table 4.8 Best performance of BADS, SPSO2011, and qABC for the NFEs obtained through 50 runs on test functions f1–f24

No.	Functions	Result of 50 runs		Result of 50 runs		Result of 50 runs	
		BADS	qABC	BADS	SPSO2011	BADS	BBA
1	Sphere (10D)	50	0	50	0	50	0
2	Rosenbrock (10D)	50	0	50	0	50	0
3	Quartic (30D)	41	9	48	2	48	2
4	Ackley (10D)	50	0	50	0	50	0
5	Schaffer (2D)	47	3	49	1	50	0
6	Easom (2D)	46	4	0	50	46	4
7	Rastrigin (10D)	50	0	50	0	50	0
8	Shekel (4D)	49	1	0	50	50	0
9	Langerman (10D)	0	50	0	50	35	15
10	Griewank (10D)	50	0	50	0	50	0
11	Branin (2D)	47	3	50	0	50	0
12	Sumpow (10D)	50	0	50	0	50	0
13	Bukin6 (2D)	19	30	0	50	2	47
14	Crossit (2D)	44	6	1	49	39	2
15	Drop (2D)	49	1	0	50	49	0
16	Shubert (2D)	20	30	0	50	50	0
17	Beale (2D)	26	24	1	49	0	50
18	McCorm (2D)	48	2	0	50	18	26
19	Camel6 (2D)	13	37	0	50	45	3

20	Boha1 (2D)	49	1	50	0	50	0
21	Colville (2D)	50	0	50	0	50	0
22	Powersum (2D)	50	0	12	38	32	18
23	Solomon (4D)	50	0	50	0	50	0
24	Alpine (2 D)	50	0	50	0	49	1
Total		BADS: 20 qABC: 4		BADS: 13 SPSO2011: 11		BADS: 21 BBA: 3	

From Table 4.9, it can easily concluded that the mean values confirm the performance figures based on NFEs. However, the standard deviations for BADS disperse more in few functions while consistently being located close to the mean in 17 functions. In general, BADS's performance was better than that of the BBA in most of the functions, with consistent performance in general.

Table 4.9 Means and standard deviations of the NFEs obtained through 50 runs for BADS and the BBA on test functions f1–f24

No.	Functions	BBA		BADS	
		Mean	Std. Dev.	Mean	Std. Dev.
1	Sphere (10D)	9180.34	2588.57	360.10	208.53
2	Rosenbrock (10D)	500042.22	5.54	276.00	0.00
3	Quartic (30)	500068.04	1.96	175511.40	158655.66
4	Ackley (10D)	500039.02	3.61	468.68	201.10
5	Schaffer (2D)	5066.22	2679.38	425.96	214.16
6	Easom (2D)	4060.24	1381.89	3153.50	10513.18
7	Rastrigin (10D)	500068.86	3.86	550.56	208.31
8	Shekel (4D)	62742.04	32020.36	1946.88	349.66
9	Langerman (10D)	385919.18	184710.55	500136.60	11.39
10	Griewank (10D)	500045.52	3.98	470.84	233.53
11	Branin (2D)	4115.46	1758.67	419.04	137.60
12	Sumpow (10D)	951.20	179.34	68.00	48.00
13	Bukin6 (2D)	500012.54	5.56	500045.52	22.54
14	Crossit (2D)	400.84	227.00	146.04	58.01
15	Drop (2D)	27392.88	30397.32	509.62	220.96
16	Shubert (2D)	460845.84	106902.97	5037.34	7030.16
17	Beale (2D)	654.84	294.37	11523.28	2290.54
18	McCorm (2D)	1025.76	451.89	1018.72	305.43
19	Camel6 (2D)	1319.68	692.52	374.34	125.12
20	Boha1 (2D)	93133.46	93331.19	465.12	187.40
21	Colville (2D)	90739.72	32563.57	1626.48	964.71
22	Powersum (2D)	74731.52	65000.37	37856.06	41498.56
23	Solomon (2D)	38934.28	39767.45	522.08	238.84
24	Alpine (2D)	3546.60	1631.45	272.48	24.64
Total		3	7	21	17

The BADS and qABC means figures in Table 4.10 reinforce the observations made in Table 4.8 about the best accuracy performance. Here, in 21 functions the standard deviations spread close to the minimum. In general, BADS outperformed qABC in terms of NFEs or speed while exhibiting more consistent performance.

Table 4.10 Means and standard deviations of NFEs obtained through 50 runs for BADS and qABC on test functions f1–f24

No.	Functions	qABC		BADS	
		Mean	Std. Dev.	Mean	Std. Dev.
1	Sphere (10D)	33446.14	9643.90	360.10	208.53
2	Rosenbrock (10D)	500048.88	25.90	276.00	0.00
3	Quartic (30)	377869.86	180710.93	175511.40	158655.66
4	Ackley (10D)	468309.60	70444.77	468.68	201.10
5	Schaffer (2D)	233117.16	235650.01	425.96	214.16
6	Easom (2D)	2650.48	4249.69	3153.50	10513.18
7	Rastrigin (10D)	180555.66	92942.62	550.56	208.31
8	Shekel (4D)	43292.62	96619.86	1946.88	349.66
9	Langerman (10D)	491350.46	60942.64	500136.60	11.39
10	Griewank (10D)	500049.58	27.24	470.84	233.53
11	Branin (2D)	1052.04	885.35	419.04	137.60
12	Sumpow (10D)	150.00	0.00	68.00	48.00
13	Bukin6 (2D)	490635.52	57616.16	500045.52	22.54
14	Crossit (2D)	274.00	115.86	146.04	58.01
15	Drop (2D)	236593.44	196388.31	509.62	220.96
16	Shubert (2D)	5662.18	9603.01	5037.34	7030.16
17	Beale (2D)	180170.24	213103.40	11523.28	2290.54
18	McCorm (2D	13572.06	17897.62	374.34	125.12
19	Camel6 (2D)	790.00	369.86	1018.72	305.43
20	Boha1 (2D)	1872.26	1145.41	465.12	187.40
21	Colville (2D)	453006.78	141528.21	1626.48	964.71
22	Powersum (2D)	463048.28	94270.83	37856.06	41498.56
23	Solomon (2D)	273926.74	238028.51	522.08	238.84
24	Alpine (2D)	2344.00	8793.92	272.48	24.64
Total		4	3	20	21

As it has been noted before, the data for SPSO2011 shown in Table 4.11 look different than the results for the other algorithms giving an indication of good performance. The means for SPSO2011, however, are aligned with those in Table 3.8, while the standard deviations looks like an improvement, with zero deviation in some functions. This can be predicted on the light of what is known about the original PSO weakness of premature convergence.

Table 4.11 Means and standard deviations of NFEs obtained through 50 runs for BADS and SPSO2011 on test functions f1–f24

No.	Functions	SPSO2011		BADS	
		Mean	Std. Dev.	Mean	Std. Dev.
1	Sphere (10D)	5884.00	437.89	360.10	208.53
2	Rosenbrock (10D)	209694.00	190711.63	276.00	0.00
3	Quartic (30D)	500000.00	0.00	175511.40	158655.66
4	Ackley (10D)	139692.00	213573.00	468.68	201.10
5	Schaffer (2D)	2122.00	544.53	425.96	214.16
6	Easom (2D)	100.00	0.00	3153.50	10513.18
7	Rastrigin (10D)	500000.00	0.00	550.56	208.31
8	Shekel (4D)	100.00	0.00	1946.88	349.66
9	Langerman (10D)	100.00	0.00	500136.60	11.39
11	Griewank (10D)	500000.00	0.00	470.84	233.53
10	Branin (2D)	500000.00	0.00	419.04	137.60
12	Sumpow (10D)	1502.00	327.10	68.00	48.00
13	Bukin6 (2D)	490634.00	65562.00	500045.52	22.54
14	Crossit (2D)	100.00	0.00	146.04	58.01
15	Drop (2D)	100.00	0.00	509.62	220.96
16	Shubert (2D)	100.00	0.00	5037.34	7030.16
17	Beale (2D)	11656.00	69765.16	11523.28	2290.54
18	McCorm (2D)	100.00	0.00	374.34	125.12
19	Camel6 (2D)	120.00	44.72	1018.72	305.43
20	Boha1 (2D)	4992.00	742.12	465.12	187.40
21	Colville (2D)	14160.00	4506.00	1626.48	964.71
22	Powersum (2D)	23094.00	23524.03	37856.06	41498.56
23	Solomon (2D)	26946.00	96573.06	522.08	238.84
24	Alpine (2D)	2912.00	1601.83	272.48	24.64
Total		10	13	14	11

The earlier discussion indicated that the proposed algorithm, BADS, outperformed the others in terms of speed with an acceptable level of stability. However, to ensure that the acquired result is statistically significant and occurred as result of introducing the new method, rather than occurring randomly, the Mann–Whitney test was again used. According to Table 4.12, which illustrates the comparison with the BBA, BADS was significantly better in terms of NFEs in 21 functions, meaning that it converged more quickly to the optimum than did the BBA. The comparison with qABC indicates significance in only 17 functions, which indicate a lower significant performance than against the BBA. Furthermore, the results in Table 4.12 indicate that BADS performed significantly better in more than half of the functions while SPSO2011's appear to be functioning higher than that of the BBA and qABC. However, as per the previous discussion, this cannot be taken as an indication of improved performance.

Table 4.12 P-values using the Mann–Whitney test (a = 0.05) for NFEs acquired by BADS over the BBA

No.	Functions	BADS–qABC		BADS–SPSO2011		BADS–BBA	
		P-value	Significant	P-value	Significant	P-value	Significant
1	Sphere	3.25E-18	Yes	2.20E-18	Yes	4.08E-18	Yes
2	Rosenbrock	3.30E-20	Yes	1.56E-20	Yes	3.21E-20	Yes
3	Quartic (30D)	4.04E-07	Yes	6.93E-15	Yes	2.05E-15	Yes
4	Ackley (10D)	4.82E-18	Yes	3.05E-18	Yes	4.60E-18	Yes
5	Schaffer (2D)	4.69E-15	Yes	4.14E-18	Yes	6.20E-18	Yes
6	Easom (2D)	3.08E-12	Yes	5.79E-21	PSO	1.58E-13	Yes
7	Rastrigin (10D)	6.45E-18	Yes	2.30E-20	Yes	6.22E-18	Yes
8	Shekel (4D)	3.64E-18	Yes	2.91E-22	PSO	2.12E-19	Yes
9	Langerman (10D)	6.99E-18	QABC	2.56E-20	PSO	5.70E-04	Yes
10	Griewank (10D)	5.43E-18	Yes	1.86E-20	Yes	5.09E-18	Yes
11	Branin (2D)	8.93E-15	Yes	1.99E-20	Yes	5.73E-18	Yes
12	Sumpow (10D)	6.78E-21	Yes	1.36E-18	Yes	1.86E-18	Yes
13	Bukin6 (2D)	3.70E-01	No	3.64E-20	PSO	1.83E-14	BBA
14	Crossit (2D)	6.71E-11	Yes	1.88E-19	PSO	8.48E-11	Yes
15	Drop (2D)	2.40E-17	Yes	2.29E-20	PSO	3.28E-17	Yes
16	Shubert (2D)	8.54E-02	No	2.57E-20	PSO	6.78E-18	Yes
17	Beale (2D)	3.39E-01	No	8.22E-17	PSO	5.71E-18	BBA
18	McCorm (2D)	2.34E-14	Yes	1.93E-20	PSO	6.46E-01	No
19	Camel6 (2D)	2.92E-05	QABC	2.61E-19	PSO	4.27E-13	Yes
20	Boha1 (2D)	6.14E-18	Yes	2.98E-18	Yes	4.39E-18	Yes
21	Colville (2D)	2.44E-17	Yes	4.77E-18	Yes	6.98E-18	Yes
22	Powersum (2D)	1.07E-17	Yes	4.02E-01	No	5.05E-04	Yes
23	Solomon (2D)	6.54E-18	Yes	4.48E-18	Yes	6.56E-18	Yes
24	Alpine (2D)	4.45E-20	Yes	3.59E-20	Yes	1.24E-18	Yes
Total		BADS: 19 qABC: 3 None: 2		BADS: 13 SPSO2011: 10 None: 1		BADS: 21 BBA: 2 None: 1	

4.6 Summary

In this chapter, a discussion about the testing of another proposed method used to improve the BA. The test was carried out on a high number of functions with a wide variety of topography and features. The results revealed that the method improved the performance of the BA in terms of accuracy in at least 70% of the functions tested. However, in terms of stability, an appropriate level was maintained. Considering that the method used is a hybridisation of two methods of sampling, this most probably prevented the algorithm from achieving a higher level of stability, as noticed for the BAwSSR algorithm examined in Chapter 3. In terms of speed, it also performed better than all other algorithms, reaching significant improvements in more than half of the total functions. However, it maintained a high SR score, close to that of the previous method, BAwSSR, gaining 91% SR. The overall conclusion from the obtained results indicate that the segmentation methods used positively affected in the enhancement of the BA.

Chapter 5 BEES ALGORITHM IMPROVEMENT USING OVERLAPPING SEGMENTATION OF SEARCH SPACE (BAOSS)

5.1 Preliminaries

In this chapter, another new method is introduced to improve the BA. As in the preceding chapters, this method was applied to the BA initialisation and global search stages. The primary technique used in this proposed algorithm was to divide the search space into overlapping segments. A discussion of the concept of dividing search space into independent segments was presented in Chapter 4. While segmentation of the domain helps diversify the samples to cover more areas of the search space, extracting one sample with all its parameters from the same segment might negatively affect performance in certain cases. This is due to some functions' parameters having different values hence being distributed throughout the search space, making it impossible to reach the optimum. This contrary to the case when all the parameters of a functions are having identical values, thus, in the previous chapter, two sampling approaches to handle these problems were used. However, the implementation of overlapping segmentations could help address this problem by allowing the same sample to fall into more than one segment with the same sampling approach. Additionally, another search technique is included to support the search capability of the proposed algorithm. The aim of this technique is to track the most promising search intervals that yield better values and, hence, guide the search to extract better optimum. It will also work to lead the search accordingly to avoid infeasible search efforts.

5.2 The Overlapping Segmentation of the Search Space Method

The core of this method is to segregate the search space of the function being optimised to a uniformly sized overlapping segment. The number of segments should be equal to or more than the stipulated initial sample parameter. The aim of this segmentation is to increase the probability that the extracted sample will be diversely located throughout the search space, while not restricting the parameters themselves to be in certain areas.

5.2.1 Search Space Overlap Segmentation

Every two consecutive segments overlap in half of their segment size. This means that the second segment starts from the centre of the first segment, which renders them as sharing half of their segment size. This process continues for the rest of the segments (Figure 5.1 & Figure 5.2). Taking the Colville function as an example, with its search space defined on the area [−10, 10], the length of the search space then will be calculated as follows:

$$L = (10 - (-10)) = 20 \tag{5.1}$$

and the length of the segment (S) is calculated as follows:

$$S = L/n \tag{5.2}$$

where n is the number of samples and L is the length of the search space. If $n = 20$, then, applying equation 5.2, S = 20/20 = 1 The general formula to calculate the segment's start and end values is the following:

if $[a_1, b_1]$ is the first segment, it will be calculated as

$$[a_1, b_1] = [a_1, b_1\text{-S}] \tag{5.3}$$

a_1 will be the starting point.

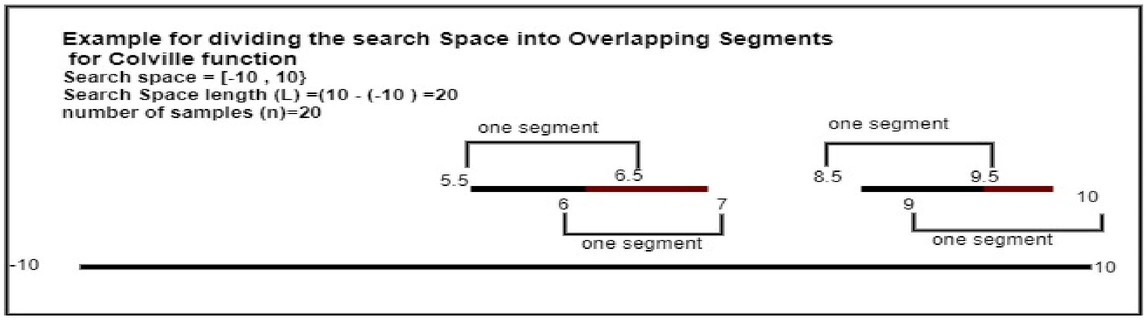
, the second segment will be calculated as calculated as

$$[a_2, b_2] = [a_1\text{-(S/2)}, b_1\text{-S}] \tag{5.4}$$

and the *n* segment will be

$$[a_n, b_n]=[a_{n-1}\text{-(S/2)}, b_{n-1}\text{-S}] \quad (5.5)$$

As in Colville functions, a_1= 10, starting from the right end of the domain, then b_1= a_1−S, which will be the interval [10, 9]. The second interval is $[a_2, b_2]$=[a_1-(S/2), b_1-S] = [9.5, 8.5], and the process continues. For a sample of segments calculated limits, see Table 5.1.


Example for dividing the search Space into Overlapping Segments for Colville function
Search space = [-10 , 10}
Search Space length (L) =(10 - (-10) =20
number of samples (n)=20
one segment
one segment
5.5
6.5
6
7
one segment
8.5
9.5
9
10
one segment
-10
10
Figure 5.1 Illustration 1 1of dividing the search space into overlapping segments for the Colville function.

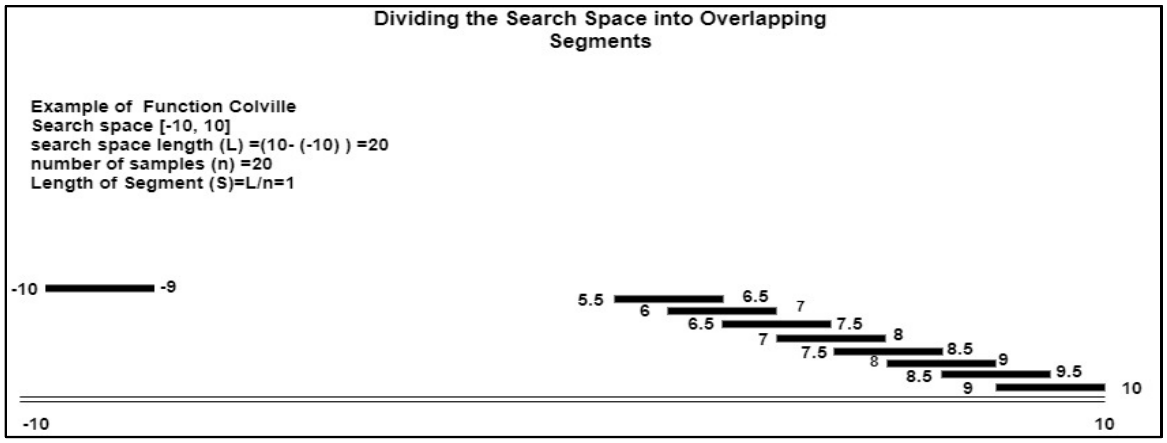

Dividing the Search Space into Overlapping Segments
Example of Function Colville
Search space [-10, 10]
search space length (L) =(10- (-10)) =20
number of samples (n) =20
Length of Segment (S)=L/n=1
-10
-9
5.5
6.5
6
7
6.5
7.5
7
8
7.5
8.5
8
9
8.5
9.5
9
10
-10
10
Figure 5.2 Illustration 2 of dividing the search space into overlapping segments for the Colville function.

Table 5.1 Sample of the calculated segment's limits

-10	-9
-	-
-	-
5.5	6.5
6	7
6.5	7.5
7	8
7,5	8.5

8	9
8.5	9.5
9	10

5.2.2 Tracking Promising Search Space Intervals

To track the promising interval of the search space in every iteration, the best population parameters are inspected for their minimum and maximum values. The search space then is restricted to these two extreme values:

- The minimum parameter value of the best population $u = min\ (x_m, n)$, and
- The maximum parameter value of the best population $v = max\ (x_m,n)$,

where m is the number of variables of a function and n is the number of samples. This will limit the search to the [v, u] interval.

5.3 Experiment Setup

The experiment to test this proposed method will follow the same performance assessment criteria used for the previously introduced algorithms. This includes accuracy value, SR, and NFE. The metrics described in Chapter 3 will be applied with the same stopping criteria in all 50 runs. Similarly, the Mann–Whitney test will be run on the results to evaluate the performance of all algorithms involved.

The same benchmark functions used with the previously proposed algorithms will also be used. Similarly, the parameter values for the BBA, SPSO2011, and qABC algorithms involved in the performance comparison will be retained (see Tables 3.3, 3.4, and 3.5 in Chapter 3). However, the proposed algorithm, BAOSS, will use different parameter values, as shown in Table 5.2.

Table 5.2 List of parameter values used for testing BAOSS

No.	Functions	n	m	nsp	e	nep	ngh	stlim

1	Sphere (10D)	100	4	10	2	30	0.03	10
2	Rosenbrock (10D)	100	4	10	2	30	0.001	10
3	Quartic(30D)	100	4	10	2	30	0.005	10
4	Ackley (10D)	100	4	10	2	30	0.0001	10
5	Schaffer (2D)	100	4	10	2	30	0.01	10
6	Easom (2D)	100	4	10	2	30	0.009	10
7	Rastrigin (10D)	100	4	10	2	30	0.0003	10
8	Shekel (4D)	100	4	10	2	30	0.001	10
9	Langerman(10D)	10	4	10	2	30	0.5	10
10	Griewank (10D)	100	4	10	2	30	0.0001	10
11	Branin (2D)	100	4	10	2	30	0.001	10
12	Sumpow (10D)	100	4	10	2	30	0.001	10
13	Bukin6 (2D)	15	4	10	2	30	0.001	10
14	Crossit (2D)	100	4	10	2	30	0.05	10
15	Drop (2D)	100	4	10	2	30	0.1	10
16	Shubert (2D)	100	4	10	2	30	0.0005	10
17	Beale (2D)	18	4	10	2	30	0.05	10
18	McCorm (2D)	100	4	10	2	30	0.05	10
19	Camel6 (2D)	20	4	10	2	30	0.005	10
20	Boha1 (2D)	150	4	10	2	30	0.05	10
21	Colville (2D)	150	4	10	2	30	0.0005	10
22	Powersum (2D)	15	4	10	2	30	0.01	10
23	Solomon (2D)	20	4	10	2	30	0.01	10
24	Alpine (2D)	100	4	10	2	30	0.01	10

5.4 Results and Discussion

5.4.1 Solution Quality (Accuracy)

The values in Table 5.3 reveal improved performance by the BAOSS. It outperformed the all the algorithms involved. The best performance was against SPSO2011, where BAOSS excelled in 23 of the 24 functions while outperforming the BBA and qABC in 21 functions. It is worth mentioning that BAOSS's performance was outstanding against the three algorithms particularly in the functions Sphere, Rosenbrock, Ackley, Griewank, Rastrigin, Quartic, Sumpow, and Colville wining most of the 50 runs. From those functions, Rosenbrock, Ackley, Griewank, and Rastrigin represent challenges to optimisation algorithms due to their multipocketed topography and complex landscapes. However, the next step is to explore other aspects of performance concerning consistency and stability.

Table 5.3 Best performance figures for the BAOSS, BBA, qABC, and SPSO2011 for accuracy values

No.	Functions	Result of 50 runs		Result of 50 runs		Result of 50 runs	
		BAOSS	qABC	BAOSS	SPSO2011	BAOSS	BBA
1	Sphere (10D)	50	0	49	1	50	0
2	Rosenbrock (10D)	50	0	50	0	50	0
3	Quartic (30D)	46	4	43	7	50	0
4	Ackley (10D)	50	0	50	0	50	0
5	Schaffer (2D)	43	7	32	18	30	20
6	Easom (2D)	34	16	50	0	40	10
7	Rastrigin (10D)	47	3	50	0	50	0
8	Shekel (4D)	31	19	50	0	47	3
9	Langerman	32	18	50	0	4	46
10	Griewank (10D)	50	0	50	0	50	0
11	Branin (2D)	16	34	0	50	39	11
12	Sumpow (10D)	48	2	50	0	50	0
13	Bukin6 (2D)	1	49	29	21	0	50
14	Crossit (2D)	41	9	50	0	37	13
15	Drop (2D)	39	11	50	0	39	11
16	Shubert (2D)	16	34	48	2	44	6
17	Beale (2D)	39	11	37	13	32	18
18	McCorm (2D)	27	23	50	0	30	20
19	Camel6 (2D)	28	22	50	0	36	14
20	Boha1 (2D)	35	15	34	16	40	10
21	Colville (2D)	48	2	46	4	50	0
22	Powersum (2D)	44	6	28	22	15	35
23	Solomon (2D)	39	11	44	6	37	13
24	Alpine (2D)	32	18	38	12	39	11
Total		BAOSS: 21		BAOSS: 23		BAOSS: 21	
		qABC: 3		SPSO2011: 1		BBA: 3	

Figures 5.3–5.11 show the performance charts in terms of accuracy by all the algorithms involved.

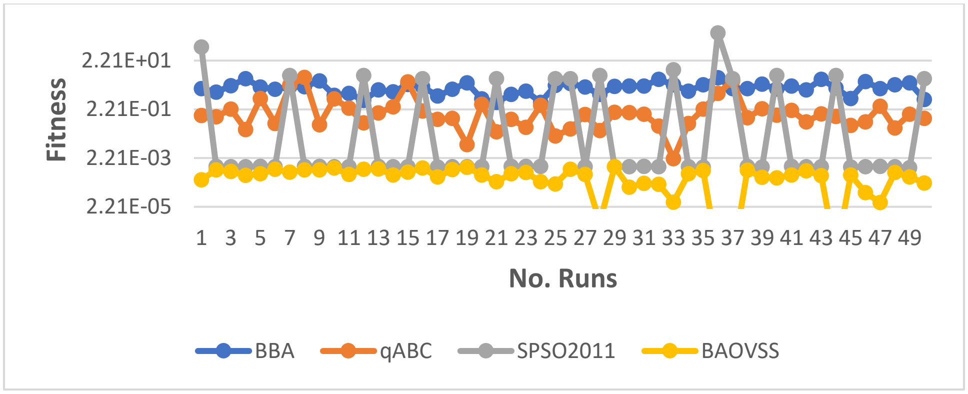


Figure 5.3 Result of 50 runs for the BBA, BAOSS, qABC, and SPSO2011 on Rosenbrock 10D function

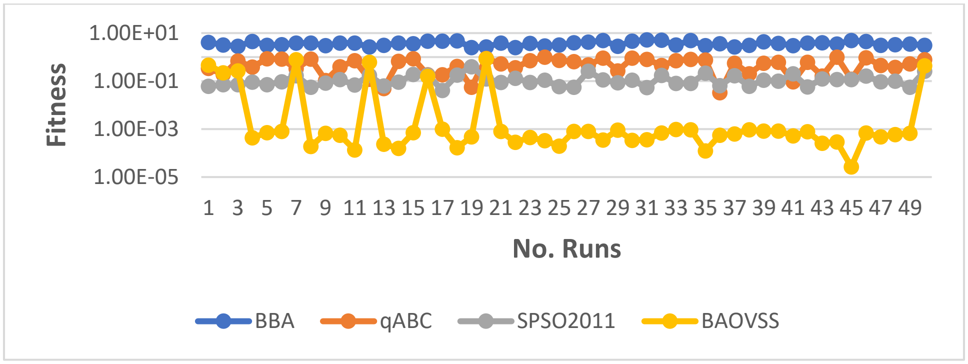


Figure 5.4 Result of 50 runs for the BBA, BAOSS, qABC, and SPSO2011 on Quartic 30D function

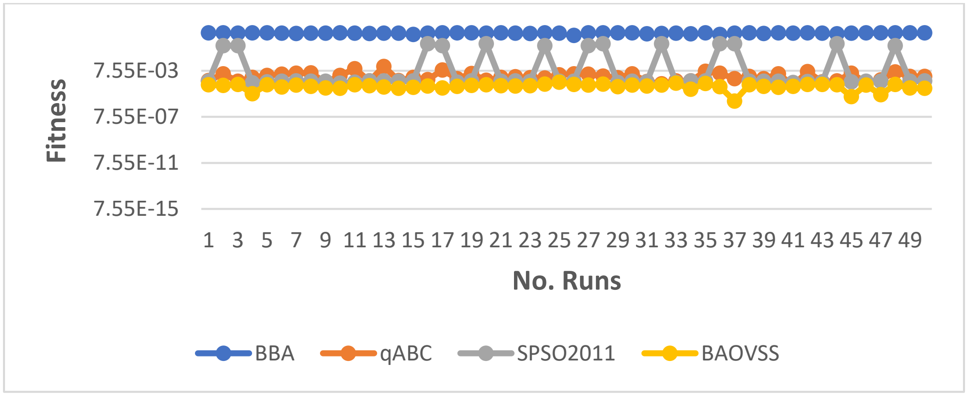


Figure 5.5 Result of 50 runs for the BBA, BAOSS, qABC, and SPSO2011 on Ackley 10D function

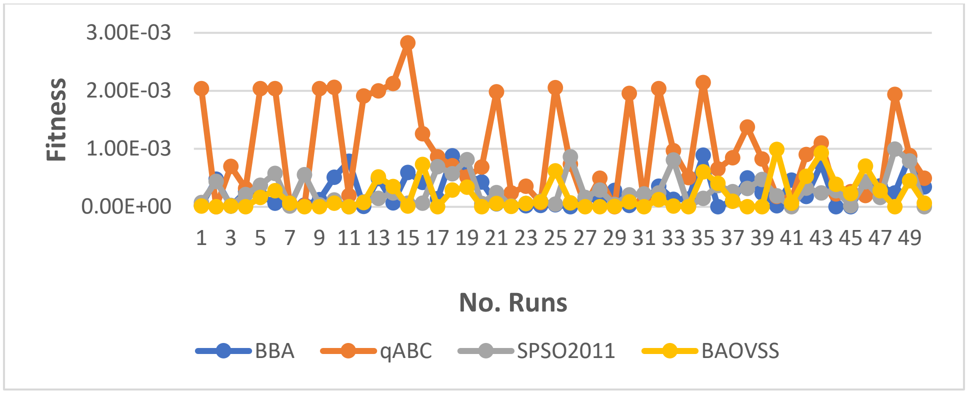


Figure 5.6 Result of 50 runs for the BBA, BAOSS, qABC, and SPSO2011 on Schaffer 2D function

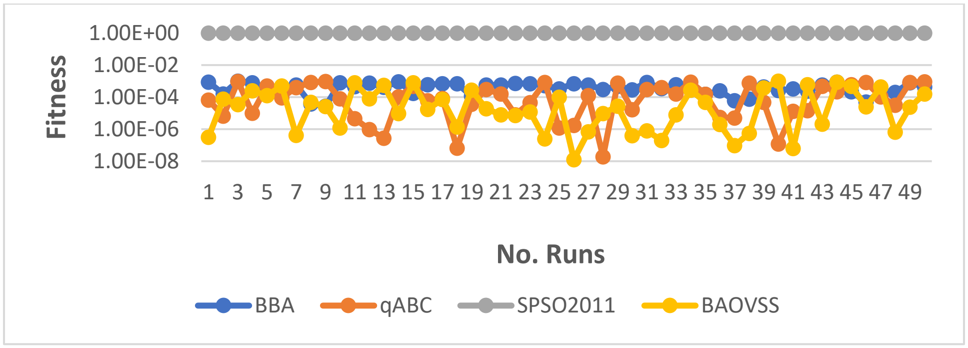


Figure 5.7 Result of 50 runs for the BBA, BAOSS, qABC, and SPSO2011 on Easom 2D function

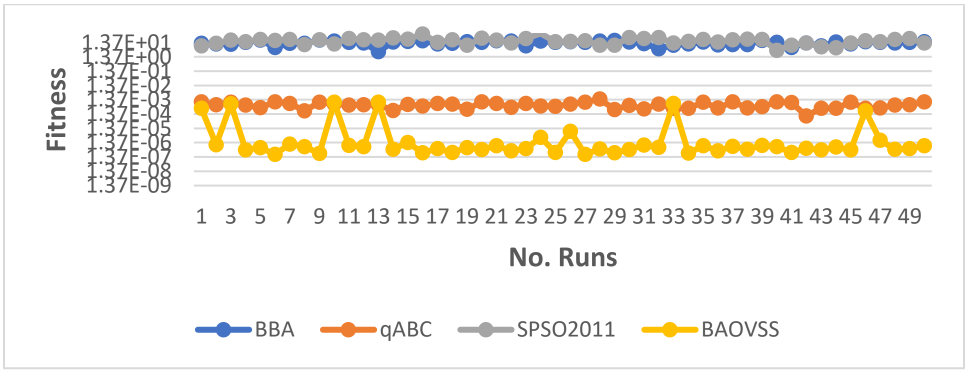


Figure 5.8 Result of 50 runs for the BBA, BAOSS, qABC, and SPSO2011 on Rastrigin 10D function

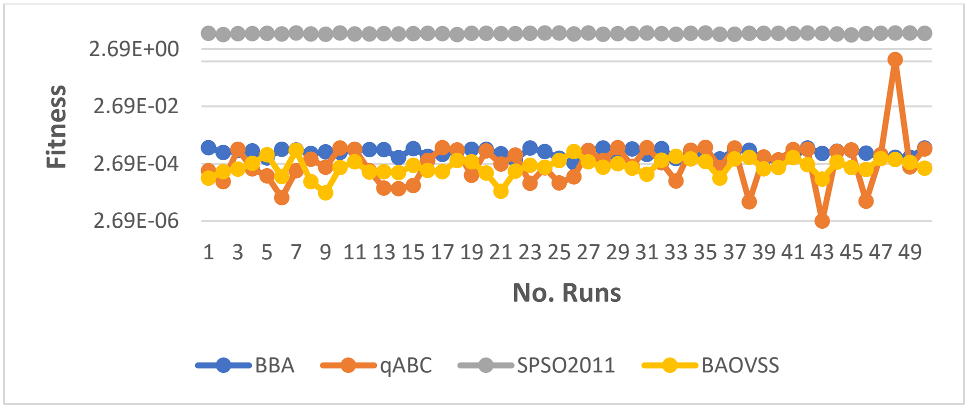


Figure 5.9 Result of 50 runs for the BBA, BAOSS, qABC, and SPSO2011 on Shekel 4D function

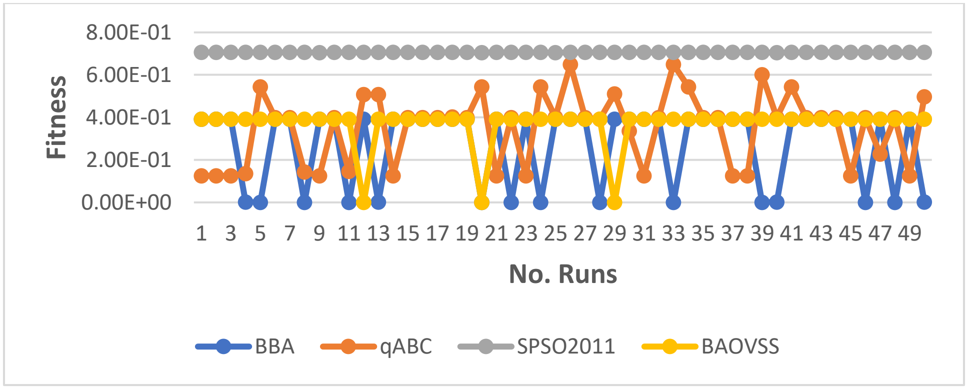


Figure 5.10 Result of 50 runs for the BBA, BAOSS, qABC, and SPSO2011 on Langerman 10D function

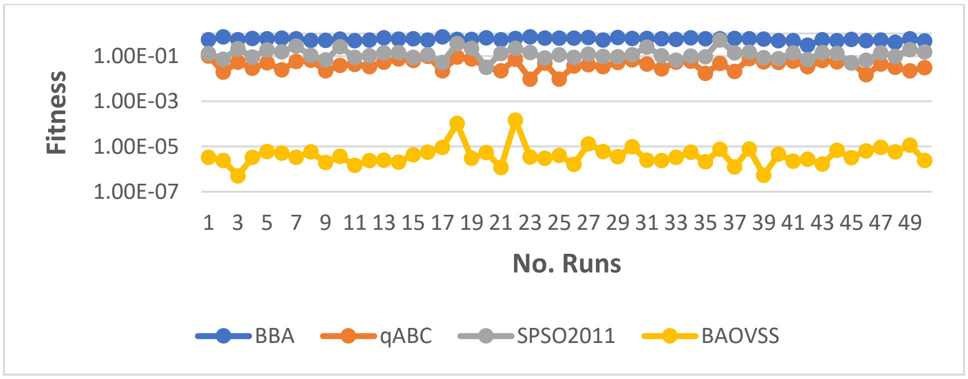


Figure 5.11 Result of 50 runs for the BBA, BAOSS, qABC, and SPSO2011 on Griewank 10D function

When exploring the means and standard deviations in Table 5.4 for BAOSS against the BBA, the means reveal that BAOSS performed better in 20 functions. The standard deviation reveals that BAOSS improved in only 17 functions indicating inconsistent performance in three functions from the best 20 of mean value. However, the Langerman, Bukin6, and Powersum functions should be excluded as BAOSS performed worse than its opponent although, the two algorithms were not able to converge in those functions. This an indication of BAOSS inability to exhibit highly stabilised performance. This might be due to applying two methods to uniformly generate samples, which causes the performance to become inconsistent.

Table 5.4 Means and standard deviations of best accuracy values for the BAOSS and BBA obtained through 50 independent runs on test functions f1–f24

No.	Functions	BBA		BAOSS	
		Mean	Std. Dev.	Mean	Std. Dev.
1	Sphere (10D)	8.19E-04	1.27E-04	2.64E-05	1.03E-04
2	Rosenbrock (10D)	1.80E+00	9.23E-01	4.60E-04	2.70E-04
3	Quartic (30D)	3.70E+00	7.34E-01	7.63E-02	1.99E-01
4	Ackley (10D)	1.38E+01	1.25E+00	3.65E-04	1.46E-04
5	Schaffer (2D)	2.57E-04	2.61E-04	1.95E-04	2.61E-04
6	Easom (2D)	4.56E-04	2.64E-04	1.62E-04	2.61E-04
7	Rastrigin (10D)	1.33E+01	3.84E+00	7.87E-05	2.31E-04
8	Shekel (4D)	6.67E-04	1.96E-04	2.56E-04	1.59E-04
9	Langerman	2.74E-01	1.79E-01	3.68E-01	9.28E-02
10	Griewank (10D)	5.68E-01	7.78E-02	9.15E-06	2.42E-05
11	Branin (2D)	6.43E-04	1.96E-04	4.09E-04	2.84E-04
12	Sumpow (10D)	6.86E-04	2.44E-04	2.64E-05	3.49E-05
13	Bukin6 (2D)	1.59E-02	4.93E-03	5.22E-02	1.41E-03
14	Crossit (2D)	4.23E-04	2.71E-04	2.06E-04	1.55E-04
15	Drop (2D)	4.06E-04	2.48E-04	1.91E-04	3.22E-04
16	Shubert (2D)	5.24E-03	4.89E-03	5.05E+00	1.71E+01
17	Beale (2D)	4.82E-04	2.74E-04	3.36E-04	2.52E-04
18	McCorm (2D)	4.87E-04	2.86E-04	2.88E-04	2.96E-04
19	Camel6 (2D)	5.26E-04	2.92E-04	4.66E-04	2.72E-04
20	Boha1 (2D)	4.68E-04	2.92E-04	2.00E-04	2.55E-04
21	Colville (2D)	9.28E-04	7.65E-05	3.30E-04	2.63E-04
22	Powersum (2D)	6.84E-04	2.26E-04	8.32E-04	1.58E-04
23	Solomon (2D)	5.49E-04	2.57E-04	2.44E-04	2.93E-04
24	Alpine (2D)	6.25E-04	2.41E-04	3.87E-04	3.34E-04
Total		4	7	20	17

Table 5.5 compares the BAOSS against qABC and reveals that the means are identical to the accuracy performance data presented in Table 5.3. BAOSS performed better in 20 functions in terms of the means, and in 21 functions in terms of the standard deviations. It worth noting that the standard deviation figures here showing unexpectedly higher values than against the

other two algorithms. Nonetheless, BAOSS's mean value for Langerman was worse than that of qABC, although it was better at finding the accurate optimum. However, neither algorithm was able to converge to the optimum within the stipulated NFE value for Langerman or Bukin6. It can be concluded that BAOSS outperformed qABC in terms of accuracy while exhibiting an adequate level of consistency.

Table 5.5 Means and standard deviations of accuracy values for the BAOSS and qABC obtained through 50 runs on test functions f1–f24

No.	Functions	qABC		BAOSS	
		Mean	Std. Dev.	Mean	Std. Dev.
1	Sphere (10D)	6.24E-04	2.49E-04	2.64E-05	1.03E-04
2	Rosenbrock 10D)	4.09E-01	8.85E-01	4.60E-04	2.70E-04
3	Quartic (30D)	5.23E-01	2.88E-01	7.63E-02	1.99E-01
4	Ackley (10D)	3.11E-03	3.05E-03	3.65E-04	1.46E-04
5	Schaffer (2D)	9.65E-04	8.04E-04	1.95E-04	2.61E-04
6	Easom (2D)	2.73E-04	3.28E-04	1.62E-04	2.61E-04
7	Rastrigin (10D)	6.22E-04	2.78E-04	7.87E-05	2.31E-04
8	Shekel (4D)	2.34E-02	1.61E-01	2.56E-04	1.59E-04
9	Langerman	3.50E-01	1.63E-01	3.68E-01	9.28E-02
10	Griewank (10D)	4.65E-02	2.19E-02	9.15E-06	2.42E-05
11	Branin (2D)	2.63E-04	2.76E-04	4.09E-04	2.84E-04
12	Sumpow (10D)	3.82E-04	1.12E-04	2.64E-05	3.49E-05
13	Bukin6 (2D)	2.81E-02	1.61E-02	5.22E-02	1.41E-03
14	Crossit (2D)	5.65E-04	3.01E-04	2.06E-04	1.55E-04
15	Drop (2D)	1.12E-02	2.17E-02	1.91E-04	3.22E-04
16	Shubert (2D)	2.69E-04	3.13E-04	5.05E+00	1.71E+01
17	Beale (2D)	1.98E-03	3.49E-03	3.36E-04	2.52E-04
18	McCorm (2D)	4.01E-04	3.26E-04	2.88E-04	2.96E-04
19	Camel6 (2D)	5.06E-04	2.83E-04	4.66E-04	2.72E-04
20	Boha1 (2D)	3.20E-04	2.78E-04	2.00E-04	2.55E-04
21	Colville (2D)	3.43E-02	3.10E-02	3.30E-04	2.63E-04
22	Powersum (2D)	7.13E-03	8.58E-03	8.32E-04	1.58E-04
23	Solomon (2D)	4.80E-02	4.88E-02	2.44E-04	2.93E-04
24	Alpine (2D)	5.58E-04	2.84E-04	3.87E-04	3.34E-04
Total		4	3	20	21

Table 5.6 shows an enhanced performance in terms of the means for BAOSS versus SPSO2011 in 22 functions whereas the standard deviation revealing an aspect of inconsistency achieving better in 17 functions only. This was noticed in five functions where the standard deviations spreading away from the means. Nonetheless, in four functions—Quartic, Easom,

Langerman, and Colville—BAOSS outperformed SPSO2011 by an enormous difference. It is important to note that the tuning of the ngh parameter positively affected the BAOSS's performance, suggesting that it was sensitive to the change in this parameter. Yet, the Mann–Whitney statistical significance test is required to validate the significance of BAOSS's improvement.

Table 5.6 Means and standard deviations of best accuracy values for the BAOSS and SPSO2011 obtained through 50 independent runs on test functions f1–f24

No.	Functions	SPSO2011		BAOSS	
		Mean	Std. Dev.	Mean	Std. Dev.
1	Sphere (10D)	7.98E-04	1.44E-04	2.64E-05	1.03E-04
2	Rosenbrock (10D)	8.88E+00	4.36E+01	4.60E-04	2.70E-04
3	Quartic (30D)	1.18E-01	6.69E-02	7.63E-02	1.99E-01
4	Ackley (10D)	3.70E-01	6.35E-01	3.65E-04	1.46E-04
5	Schaffer (2D)	2.69E-04	2.60E-04	1.95E-04	2.61E-04
6	Easom (2D)	1.00E+00	2.74E-06	1.62E-04	2.61E-04
7	Rastrigin (10D)	1.93E+01	1.01E+01	7.87E-05	2.31E-04
8	Shekel (4D)	9.32E+00	3.44E-01	2.56E-04	1.59E-04
9	Langerman	7.05E-01	9.64E-05	3.68E-01	9.28E-02
10	Griewank (10D)	1.38E-01	8.27E-02	9.15E-06	2.42E-05
11	Branin (2D)	3.58E-07	0.00E+00	4.09E-04	2.84E-04
12	Sumpow (10D)	5.48E-04	2.73E-04	2.64E-05	3.49E-05
13	Bukin6 (2D)	7.35E-02	3.97E-02	5.22E-02	1.41E-03
14	Crossit (2D)	4.43E-02	4.49E-02	2.06E-04	1.55E-04
15	Drop (2D)	2.25E-01	1.14E-01	1.91E-04	3.22E-04
16	Shubert (2D)	7.88E+01	4.48E+01	5.05E+00	1.71E+01
17	Beale (2D)	1.57E-02	1.07E-01	3.36E-04	2.52E-04
18	McCorm (2D)	1.58E-01	1.32E-01	2.88E-04	2.96E-04
19	Camel6 (2D)	4.44E-01	2.85E-01	4.66E-04	2.72E-04
20	Boha1 (2D)	4.12E-04	2.65E-04	2.00E-04	2.55E-04
21	Colville (2D)	8.08E-04	1.76E-04	3.30E-04	2.63E-04
22	Powersum (2D)	8.31E-04	2.21E-04	8.32E-04	1.58E-04
23	Solomon (2D)	2.64E-03	1.39E-02	2.44E-04	2.93E-04
24	Alpine (2D)	6.69E-04	2.35E-04	3.87E-04	3.34E-04
Total:		2	7	22	17

Table 5.7 presents the p-values obtained using the Mann–Whitney test for all the algorithms involved in this experiment. According to the results, BAOSS exhibited noteworthy performance in achieving accurate optima against all other algorithms. The comparisons of BAOSS against qABC, BBA, and SPSO2011 show that BAOSS significantly outperformed

them in 18, 20, and 22 functions, respectively. In total, there was a significant improvement by BAOSS in at least two-thirds of the functions.

Table 5.7 P-values using the Mann–Whitney test (a = 0.05) for accuracy acquired by BAOSS over qABC, BBA, and SPSO2011

No.	Functions	BAOSS–qABC		BAOSS–SPSO2011		BAOSS–BBA	
		P-value	Significant	P-value	Significant	P-value	Significant
1	Sphere (10 D)	4.21E-17	Yes	1.84E-17	Yes	1.63E-17	Yes
2	Rosenbrock (10D)	7.07E-18	Yes	1.07E-17	Yes	7.07E-18	Yes
3	Quartic (30D)	1.85E-13	Yes	2.32E-09	Yes	7.07E-18	Yes
4	Ackley (10D)	3.32E-17	Yes	9.35E-18	Yes	7.07E-18	Yes
5	Schaffer (2D)	1.32E-08	Yes	2.07E-02	Yes	3.96E-02	Yes
6	Easom (2D)	2.81E-02	Yes	9.12E-20	Yes	1.74E-08	Yes
7	Rastrigin (10D)	4.51E-14	Yes	7.02E-18	Yes	7.07E-18	Yes
8	Shekel (4D)	1.23E-01	No	7.07E-18	Yes	1.92E-14	Yes
9	Langerman (10D)	5.41E-03	Yes	7.07E-18	Yes	8.97E-13	BBA
10	Griewank (10D)	7.07E-18	Yes	7.07E-18	Yes	7.07E-18	Yes
11	Branin (2D)	2.68E-03	qABC	3.31E-20	SPSO2011	7.48E-05	Yes
12	Sumpow (10D)	6.48E-17	Yes	1.21E-17	Yes	7.07E-18	Yes
13	Bukin6 (2D)	7.49E-16	qABC	1.69E-01	No	7.07E-18	BBA
14	Crossit (2D)	1.36E-07	Yes	7.07E-18	Yes	4.55E-05	Yes
15	Drop (2D)	2.30E-08	Yes	7.07E-18	Yes	1.69E-06	Yes
16	Shubert (2D)	2.23E-03	qABC	5.97E-16	Yes	7.95E-10	Yes
17	Beale (2D)	2.14E-06	Yes	1.15E-03	Yes	6.81E-03	Yes
18	McCorm (2D)	1.36E-01	No	7.07E-18	Yes	4.33E-04	Yes
19	Camel6 (2D)	5.24E-01	No	7.07E-18	Yes	3.57E-01	No
20	Boha1 (2D)	4.27E-03	Yes	2.14E-05	Yes	1.32E-06	Yes
21	Colville (2D)	3.51E-17	Yes	4.42E-13	Yes	7.12E-17	Yes
22	Powersum (2D)	1.72E-12	Yes	1.57E-01	No	6.04E-04	BBA
23	Solomon (4D)	2.34E-08	Yes	2.23E-09	Yes	1.26E-11	Yes
24	Alpine (2D)	1.87E-06	Yes	4.36E-09	Yes	2.69E-04	Yes
Total		BAOSS: 18 qABC: 3 None:3		BAOSS: 21 SPSO2011: None:3		BAOSS: 20 BBA: 3 None:1	

5.4.2 SR and NFEs

Table 5.8 indicates that the SRs for BAOSS are far higher than those of all other algorithms examined in this experiment demonstrating robust behaviour. Although BAOSS could not achieve 100% SR convergence in four functions, the achieved SR is almost the same as that achieved by the other two proposed algorithms: BAwSSR and BADS. Function Bukin6 is the most difficult for all the algorithms involved in this research, with an SR of 0% reached by the

BBA and BAOSS, 4% achieved by qABC and SPSO2011. Langerman also a difficult function, with qABC, BAOSS, and BBA achieving SR values of 2%, 6%, and 30%, respectively. Although SPSO2011 achieved an SR of 100% in Langerman, having this value for Langerman and for some other functions cannot be considered an indication of enhanced performance, whereas SPSO2011 could not come any closer to the optimum due to suffering from premature convergence.

Table 5.8 SR of the BAOSS compared with BBA, SPSO2011, and qABC based on NFEs obtained through 50 runs on test functions f1–f24

No.	Functions	qABC success rate	SPSO2011 success rate	BBA success rate	BAOSS success rate
1	Sphere (10D)	100%	100%	100%	100%
2	Rosenbrock (10D)	0%	74%	0%	100%
3	Quartic (30D)	36%	0%	0%	84%
4	Ackley (10D)	22%	74%	0%	100%
5	Schaffer (2D)	64%	100%	100%	100%
6	Easom (2D)	100%	100%	100%	100%
7	Rastrigin (10D)	98%	0%	0%	100%
8	Shekel (4D)	98%	100%	100%	100%
9	Langerman (10D)	2%	100%	30%	6%
10	Griewank (10D)	0%	0%	0%	100%
11	Branin (2D)	100%	0%	100%	100%
12	Sumpow (10D)	100%	100%	100%	100%
13	Bukin6 (2D)	4%	4%	0%	0%
14	Crossit (2D)	100%	100%	100%	100%
15	Drop (2D)	74%	100%	100%	100%
16	Shubert (2D)	100%	100%	14%	92%
17	Beale (2D)	76%	98%	100%	100%
18	McCorm (2D)	100%	100%	100%	100%
19	Camel6 (2D)	100%	100%	100%	100%
20	Boha1 (2D)	100%	100%	100%	100%
21	Colville (2D)	10%	100%	100%	100%
22	Powersum (2D)	16%	100%	100%	100%
23	Solomon (4D)	48%	96%	100%	100%
24	Alpine (2D)	100%	100%	100%	100%
SR Average		65%	77%	69%	91%

Table 5.9 demonstrates the NFE figures acquired from evaluating the performance of the BAOSS algorithm. The figures for the BBA, qABC, and SPSO2011 versus BAOSS are 20, 17, and 14, respectively which means that the best performance was against the BBA, followed by the qABC and then SPSO2011. The results suggest that BAOSS was not particularly fast against

qABC and then SPSO2011 algorithms although it attained an improved performance. Nonetheless, as noted before, the SPSO2011 figures is not indication of an improved performance. However, it is essential to look at other performance statistics to evaluate the algorithm's behaviour consistency.

Table 5.9 Best performance of BAOSS, SPSO2011, and qABC for NFEs obtained through 50 independent runs on test functions f1–f24

No.	Functions	Result of 50 runs		Result of 50 runs		Result of 50 runs	
		BAOSS	qABC	BAOSS	SPSO2011	BAOSS	BBA
1	Sphere (10 D)	50	0	50	0	50	0
2	Rosenbrock (10D)	50	0	50	0	50	0
3	Quartic(30D)	30	20	42	8	42	8
4	Ackley (10D)	50	0	49	1	50	0
5	Schaffer (2D)	45	5	50	0	50	0
6	Easom (2D)	39	11	0	50	49	1
7	Rastrigin (10D)	50	0	50	0	50	0
8	Shekel (4D)	50	0	0	50	50	0
9	Langerman (10D)	37	13	0	50	36	14
10	Griewank (10D)	50	0	50	0	50	0
11	Branin (2D)	7	43	50	0	45	5
12	Sumpow(10D)	0	50	50	0	0	50
13	Bukin6 (2D)	21	29	0	49	8	`41
14	Crossit (2D)	16	34	0	50	33	17
15	Drop (2D)	49	1	0	49	49	1
16	Shubert (2D)	2	48	0	50	49	1
17	Beale (2D)	48	2	46	4	11	39
18	McCorm (2D)	48	12	0	50	42	8
19	Camel6 (2D)	0	50	0	50	1	49
20	Boha1 (2D)	24	26	40	10	49	1
21	Colville (2D)	49	1	49	1	50	0
22	Powersum (2D)	50	0	12	38	29	21
23	Solomon (4D)	47	3	46	4	47	3
24	Alpine (2 D)	49	1	50	0	49	1
Total		BAOSS: 17	qABC: 7	BAOSS: 14	SPSO2011: 10	BAOSS: 20	BBA: 4

The data in Tables 5.10, 5.11, and 5.12 show the means and standard deviations of NFE values for the algorithms involved. Obviously, these data conform with those for best performance (Table 5.9) getting 17,13 and 20 for qABC, SPSO2011 and BBA, respectively. BAOSS displayed its best performance against the BBA (Table 5.10). However, the results demonstrate a gap between the means and standard deviations against BBA with values of 20 for the mean and 16 for standard deviation, which means the generated values spreading away from the mean. In general, the standard deviations for BAOSS appear to be particularly low obtaining values of 17, 13 and 16 respectively, indicating inconsistent performance. To evaluate the significance of BAOSS's improvement, the Mann–Whitney statistical significance test was conducted (Table 5.13).

Table 5.10 Means and standard deviations of the NFEs obtained through 50 independent runs for BAOSS and BBA on test functions f1–f24

No.	Functions	BBA		BAOSS	
		Mean	Std. Dev.	Mean	Std. Dev.
1	Sphere (10D)	9180.34	2588.57	452.00	0.00
2	Rosenbrock (10D)	500042.22	5.54	511.84	204.07
3	Quartic (30)	500068.04	1.96	228396.52	169464.98
4	Ackley (10D)	500039.02	3.61	1099.68	2052.33
5	Schaffer (2D)	5066.22	2679.38	557.60	308.88
6	Easom (2D)	4060.24	1381.89	853.28	649.09
7	Rastrigin (10D)	500068.86	3.86	617.44	73.92
8	Shekel (4D)	62742.04	32020.36	508.32	187.72
9	Langerman (10D)	385919.18	184710.55	471314.88	113721.09
10	Griewank (10D)	500045.52	3.98	635.04	49.28
11	Branin (2D)	4115.46	1758.67	1356.64	238.76
12	Sumpow (10D)	951.20	179.34	276.00	0.00
13	Bukin6 (2D)	500012.54	5.56	500047.38	25.40
14	Crossit (2D)	400.84	227.00	276.00	0.00
15	Drop (2D)	27392.88	30397.32	1575.08	1515.25
16	Shubert (2D)	460845.84	106902.97	101874.72	153655.38
17	Beale (2D)	654.84	294.37	1010.64	388.56
18	McCorm (2D	1025.76	451.89	3412.32	380.24
19	Camel6 (2D)	1319.68	692.52	521.12	141.47
20	Boha1 (2D)	93133.46	93331.19	7839.68	22692.12
21	Colville (2D)	90739.72	32563.57	2256.32	2540.24
22	Powersum (2D)	74731.52	65000.37	47143.56	39059.28
23	Solomon (2D)	38934.28	39767.45	2528.50	2726.19
24	Alpine (2D)	3546.60	1631.45	395.68	102.26
Total		4	8	20	16

Table 5.11 Means and standard deviations of NFEs obtained through 50 runs for BAOSS and qABC on test functions f1–f24

No.	Functions	qABC		BAOSS	
		Mean	Std. Dev.	Mean	Std. Dev.
1	Sphere (10D)	33446.14	9643.90	452.00	0.00
2	Rosenbrock (10D)	500048.88	25.90	511.84	204.07
3	Quartic (30)	377869.86	180710.93	228396.52	169464.98
4	Ackley (10D)	468309.60	70444.77	1099.68	2052.33
5	Schaffer (2D)	233117.16	235650.01	557.60	308.88
6	Easom (2D)	2650.48	4249.69	853.28	649.09

7	Rastrigin (10D)	180555.66	92942.62	617.44	73.92
8	Shekel (4D)	43292.62	96619.86	508.32	187.72
9	Langerman (10D)	491350.46	60942.64	471314.88	113721.09
10	Griewank (10D)	500049.58	27.24	635.04	49.28
11	Branin (2D)	1052.04	885.35	1356.64	238.76
12	Sumpow (10D)	150.00	0.00	276.00	0.00
13	Bukin6 (2D)	490635.52	57616.16	500047.38	25.40
14	Crossit (2D)	274.00	115.86	276.00	0.00
15	Drop (2D)	236593.44	196388.31	1575.08	1515.25
16	Shubert (2D)	5662.18	9603.01	101874.72	153655.38
17	Beale (2D)	180170.24	213103.40	1010.64	388.56
18	McCorm (2D)	790.00	369.86	3412.32	380.24
19	Camel6 (2D)	13572.06	17897.62	521.12	141.47
20	Boha1 (2D)	1872.26	1145.41	7839.68	22692.12
21	Colville (2D)	453006.78	141528.21	2256.32	2540.24
22	Powersum (2D)	463048.28	94270.83	47143.56	39059.28
23	Solomon (2D)	273926.74	238028.51	2528.50	2726.19
24	Alpine (2D)	2344.00	8793.92	430.88	114.82
Total		7	6	17	17

Table 5.12 Means and standard deviations of NFEs obtained through 50 runs for BADS and SPSO2011 on test functions f1–f24

No.	Functions	SPSO2011		BAOSS	
		Mean	Std. Dev.	Mean	Std. Dev.
1	Sphere (10D)	5884.00	437.89	452.00	0.00
2	Rosenbrock (10D)	209694.00	190711.63	511.84	204.07
3	Quartic (30D)	500000.00	0.00	228396.52	169464.98
4	Ackley (10D)	139692.00	213573.00	1099.68	2052.33
5	Schaffer (2D)	2122.00	544.53	557.60	308.88
6	Easom (2D)	100.00	0.00	853.28	649.09
7	Rastrigin (10D)	500000.00	0.00	617.44	73.92
8	Shekel (4D)	100.00	0.00	508.32	187.72
9	Langerman (10D)	100.00	0.00	471314.88	113721.09
10	Griewank (10D)	500000.00	0.00	635.04	49.28
11	Branin (2D)	500000.00	0.00	1356.64	238.76
12	Sumpow (10D)	1502.00	327.10	276.00	0.00
13	Bukin6 (2D)	490634.00	65562.00	500047.38	25.40
14	Crossit (2D)	100.00	0.00	276.00	0.00
15	Drop (2D)	100.00	0.00	1575.08	1515.25
16	Shubert (2D)	100.00	0.00	101874.72	153655.38
17	Beale (2D)	11656.00	69765.16	1010.64	388.56
18	McCorm (2D)	100.00	0.00	521.12	141.47
19	Camel6 (2D)	120.00	44.72	3412.32	380.24
20	Boha1 (2D)	4992.00	742.12	7839.68	22692.12
21	Colville (2D)	14160.00	4506.00	2256.32	2540.24
22	Powersum (2D)	23094.00	23524.03	47143.56	39059.28
23	Solomon (2D)	26946.00	96573.06	2528.50	2726.19
24	Alpine (2D)	2912.00	1601.83	430.88	114.82
Total		11	13	13	10

The p-values in Table 5.13 match with the findings from examining the data on NFE best performance in Table 5.9. Specifically, the performance by BAOSS was significantly better than that of the BBA in 21 functions and that of the qABC and SPSO2011 in 17 and 14 functions, respectively. While this reflects satisfactory performance against the BBA, it did not achieve the same level of excellence against the qABC. In general, the figures indicated an acceptable convergence speed to find the optimum suggesting that the method used positively impacted BAOSS for good performance.

Table 5.13 P-values using the Mann–Whitney test (a = 0.05) for NFEs acquired by BAOSS over BBA

No.	Functions	BAOSS–qABC		BAOSS–SPSO2011		BAOSS–BBA	
		P-values	Significant	P-values	Significant	P-values	Significant
1	Sphere (10D)	3.31E-20	Yes	1.72E-20	Yes	3.28E-20	Yes
2	Rosenbrock (10D)	5.25E-19	Yes	3.40E-19	Yes	5.11E-19	Yes
3	Quartic (30)	2.08E-03	Yes	2.35E-10	Yes	4.45E-09	Yes
4	Ackley (10D)	1.23E-19	Yes	8.62E-19	Yes	1.16E-19	Yes
5	Schaffer(2D)	4.56E-14	Yes	9.32E-18	Yes	5.03E-18	Yes
6	Easom (2D)	1.23E-06	Yes	1.20E-20	SPSO2011	2.23E-17	Yes
7	Rastrigin (10D)	2.13E-19	Yes	2.92E-22	Yes	2.04E-19	Yes
8	Shekel (4D)	1.75E-19	Yes	2.08E-22	SPSO2011	1.63E-19	Yes
9	Langerman (10D)	2.60E-06	Yes	2.51E-20	SPSO2011	2.03E-04	Yes
10	Griewank (10D)	4.72E-20	Yes	4.15E-23	Yes	4.35E-20	Yes
11	Branin (2D)	8.97E-11	qABC	1.90E-20	Yes	5.31E-13	Yes
12	Sumpow (10D)	2.63E-23	qABC	1.61E-20	Yes	2.84E-20	Yes
13	Bukin6 (2D)	3.34E-01	No	1.33E-19	SPSO2011	1.52E-09	BBA
14	Crossit (2D)	8.44E-04	qABC	1.59E-23	SPSO2011	3.19E-03	Yes
15	Drop (2D)	1.51E-15	Yes	7.31E-20	SPSO2011	1.23E-14	Yes
16	Shubert (2D)	8.41E-17	qABC	2.58E-20	SPSO2011	2.75E-16	Yes
17	Beale (2D)	3.55E-14	Yes	3.20E-10	Yes	5.27E-06	BBA
18	McCorm (2D)	5.60E-18	qABC	2.02E-20	SPSO2011	6.18E-18	BBA
19	Camel6 (2D)	6.75E-13	Yes	2.41E-19	SPSO2011	3.53E-10	Yes
20	Boha1 (2D)	4.83E-01	No	2.90E-06	Yes	3.40E-14	Yes
21	Colville (2D)	1.77E-17	Yes	1.01E-17	Yes	4.17E-18	Yes
22	Powersum (2D)	1.01E-17	Yes	2.28E-05	SPSO2011	4.75E-02	Yes
23	Solomon (2D)	1.41E-14	Yes	9.75E-13	Yes	5.99E-13	Yes
24	Alpine (2D)	3.19E-17	Yes	1.08E-18	Yes	3.75E-17	Yes
Total		BAOSS: 17 qABC: 5 None: 2		BAOSS: 14 SPSO2011: 10		BAOSS: 21 BBA: 3	

5.5 Summary

This chapter has introduced a new method based on partitioning the search space into overlapping segments to help generate samples evenly while permitting the diversification of parameter values to accommodate different types of problems to optimise the sampling process. These techniques were applied to the initialisation and global search stages. They were tested on 24 functions with a wide variety of topography and complexity traits. The experimental results of the proposed algorithm, BAOSS, successfully indicated a level of accuracy in 75% of the functions while maintaining acceptable performance in terms of speed.

Chapter 6 APPLICATIONS

6.1 Single-Objective Functions Without Constraints

The proposed algorithms were tested on the gear train problem to evaluate how it is performing on engineering problems. The gear train design problem is an engineering problem that aims to minimise the gear ratio to be particularly close to 1/6.931 (Parsopoulos and Vrahatis, 2005). The design variables used here represent the number of teeth for every gear, restricted to values between 12 and 60.

To reduce the error value, the gear ratio should be as close as possible to 1/6.931 (Kannan and Kramer, 1994). To calculate the error value:

$$\text{Error value (\%)} = \frac{Gear\ Ratio - \left({}^{1}/_{6.931}\right)}{\left({}^{1}/_{6.931}\right)} \times 100\% \tag{6.1}$$

$$\text{Gear ratio} = \frac{x_3 x_2}{x_1 x_4} \tag{6.2}$$

For details about the gear train problem, its objective function, and its diagram, see Appendix A.

The results of the experiment were obtained through 20 runs, and the algorithm was forced to conduct a minimum of 100 iterations to acquire a good result. The stopping criteria used are the same as those used to test the algorithm with benchmark functions, or 500,000 NFEs. The selection of parameter values for the proposed algorithms, as listed in Table 6.1, was partially decided through trial and error and partially from the literature review of other algorithms that have attempted to improve the BA. The results obtained by the test were compared with some of those found in the literature for the ABC (Akay and Karaboga, 2010), CS (Gandomi *et al.*, 2013), and PSO–GA (Garg, 2016).

Table 6.1 Gear train problem parameters

Parameter	Parameter value		
	BAwSSR	BADS	BAOSS
Number of scout bees, *n*	20	20	20
Number of selected bees, *m*	4	4	4
Number of elite bees, *e*	2	2	2
Number of forager bees for elite bee, *nep*	30	30	30
Number of foragers for selected bee, *nsp*	10	10	10
Neighbourhood size, *ngh*	0.1	0.1	0.1

The data in Tables 6.2 and 6.3 present the results of testing the qABC, SPSO2011, BBA, BAwSSR, BADS, BAOSS, PSO–GA, ABC, and CS on the gear train problem. The best solution was achieved by PSO–GA, BAwSSR, ABC, and CS. They obtained a gear train value of 0.14428 with an error rate of 0.001%. This was among the best results found when reviewing the literature on solving the gear train problem. Nevertheless, according to the means and standard deviations, BAwSSR in general did not perform as consistently and robustly as PSO–GA, ABC, and CS. Yet, this result is an indication of the ability of the BAwSSR algorithm to improve its performance. On the other hand, the qABC and SPSO2011 as well as BADS and BAOSS, acquired the lowest solution values among all algorithms involved. Likewise, BADS and BAOSS acquired error values of 0.5% and 0.12%, respectively, which are the lowest in the group. The conclusion is that the proposed method used in the BAwSSR algorithm was more effective at optimising the gear train parameters than were the methods used in BADS and BAOSS. Moreover, regarding speed, it can be stated that ABC was the fastest, as it needed only 60 NFEs to achieve its best result, which is the lowest among all, while CS and BAwSSR needed 5,000 and 14,555 function evaluations, respectively.

Table 6.2 Performance figures comparing the qABC, SPSO2011, BBA, BAwSSR, BADS, and BAOSS on the gear train problem.

Item	qABC	SPS02011	BBA	BAwSSR	BADS	BAOSS
Mean	0.0001642317	0.0001754215	1.580352 E-05	6.12116 E-05	1.934146E-05	3.632972E-05
Std. Dev	0.0002518493	0.000262571	1.874908 E-05	7.858313 E-05	3.236422E-05	1.52026E-05
Best solution	2.357641 E-09	1.11729 E-09	9.745653 E-10	2.700857E-12	6.654886 E-09	3.064532E-08
X1	48	51	47	49	35	43
X2	12	15	24	19	16	19
X3	30	27	13	16	12	17
X4	52	55	46	43	38	52
Gear ratio	0.1442308	0.144385	0.1443108	0.144281	0.1443609	0.1444544
Error %	0.004%	0.01%	0.003%	0.001%	0.5%	0.12%
Evaluations	150	100	14,526	14,555	9,725	14,542

Table 6.3 Performance figures for the ABC, PSO–GA, CS, BAwSSR, BADS, and BAOSS on the gear train problem

Item	ABC	PSO–GA	CS	BAwSSR	BADS	BAOSS
Mean	3.641339 E-10	1.2149 E- 10	1.9841E-9	6.12116e-05	1.934146E-05	3.632972E-05
Std. Dev.	5.525811E-10	8.7787 E-10	3.5546 E-9	7.858313 E-05	3.236422E-05	1.52026E-05
Best solution	2.700857 E-12	2.70085 E-12	2.7009 E-12	2.700857E-12	6.654886E09	3.064532E-08
X1	49	43	43	49	35	43
X2	16	16	16	19	16	19
X3	19	19	19	16	12	17
X4	43	49	49	43	38	52
Gear ratio	0.144281	0.14428096	0.144281	0.144281	0.1443609	0.1444544
Error %	0.001%	0.001%	0.001%	0.001%	0.5%	0.12%
Evaluations	60	NA	5,000	14,555	9,725	14,542

6.2 Single-Objective Functions with Constraints

The engineering problem selected to apply the proposed algorithms here is the tension/compression spring, a single-objective function with constraints. The aim of this problem is to reduce the spring steel wire volume, which means decreasing the weight of the spring (Guo, 2004). More details on tension/compression springs are included in Appendix B. Engineering design problems are predominantly complex and nonlinear, with many variables and constraints that must be satisfied to achieve an optimised solution (Gandomi *et al.*, 2013).

To ensure the variables were restricted to the domain of the values, a check point was implemented to ensure that the variables' values located within the limit of their domains and to regenerate them if they were located outside their domains. With regards to the constraints, at the end of every iteration, only the feasible solutions that satisfied all the constraints were considered. One hundred runs were performed, and every run was forced to achieve at least 100 iterations. To evaluate the performance of the three proposed algorithms, firstly they were compared with the qABC, SPSO2011, and BBA, which is the set of algorithms used for testing benchmark problems and using the same parameter settings applied previously. Next, their performance was compared with a collection of algorithms that were applied successfully on the tension/compression spring, as selected from the literature. They are the following: evolutionary strategies (($\mu + \lambda$)−ES; Mezura-Montes and Coello, 2005), PSO (He *et al.*, 2004), ABC (Akay and Karaboga, 2010), society and civilisation algorithm (SCA; Ray and Liew, 2003), and unified particle swarm optimisation (UPSOm; Parsopoulos and Vrahatis, 2005). Table 6.3 shows the parameters used to apply the proposed algorithms on the tension/compression spring.

Table 6.4 Tension/compression spring parameter values

Algorithm	n	m	nsp	e	nep	ngh
BAwSSR	100	4	10	2	30	0.0001
BADS	20	4	10	2	30	0.001
BAOSS	20	4	10	2	30	0.001

Tables 6.5 and 6.6 show the values acquired by all three algorithms applied to the tension/compression spring as well as the other algorithms involved. The figures reveal that the BBA achieved the best minimum value for the weight of tension/compression springs compared to the other algorithms. BAwSSR comes next while ABC, SCA, ($\mu + \lambda$)−ES, and SPSO2011 exhibited similar performance. However, some algorithms—qABc, UPSOm, BADS, and BAOSS—exhibited poor performance in general.

The means and standard deviations in Tables 6.5 and 6.6 reveal that the BBA and BAwSSR could not maintain their enhanced performance in all the runs, although the BBA was the fastest in terms of NFEs, with 8,099 function evaluations. Nevertheless, PSO exhibited the best performance with great accuracy and stable performance. Generally, BAwSSR performed better than BADS and BAOSS, which suggests that the method used to enhance it might be more efficient than the method used on the other two. Although BAwSSR could not outperform the original BBA algorithm, the above findings imply an acceptable performance by BAwSSR and its potential if further enhancements are applied.

Table 6.5 Performance figures comparison for the qABC, SPSO2011, BBA, BAwSSR, BADS, and BAOSS on the tension/compression spring problem

Tension/ compression spring	qABC	SPSO2011	BBA	BAwSSR	BADS	BAOSS
Best solution	0.012903 96	0.01266856	0.0061625	0.008838	0.0147636	0.0151
Mean	0.067087	0.01274005	0.01378989	0.035154	0.1605027	0.055040
Std. Dev	0.044828	0.00021	0.013025	0.012017	0.0852746	0.053895
Evaluations	23,661	N/A	8,099	17,889	9,719	10,684

Table 6.6 Comparison of the figures of group of competitor algorithms with the proposed algorithms on the tension/compression spring problem for each algorithm

Tension/ compression spring	ABC	SCA	(μ + λ)-ES	UPSOm	PSO	BAwSSR	BADS	BAOS S
Best solution	0.012665	0.012669	0.012689	0.0131200	0.012665	0.008838	0.0147636	0.0151
Mean	0.012709	0.012923	0.013165	0.0229478	0.012702	0.035154	0.1605027	0.0550 4
Std. Dev.	0.012813	0.00059	0.00039	0.0072	0.000041	0.012017	0.053895	0.0553
Evaluations	30,000	25,167	30,000	100,000	15,000	17,889	9,719	10,684

Table 6.7 List of variable and constraint values achieved by the BAwSSR, BADS, and BAOSS algorithms

Variables and constraints	BAwSSR	BADS	BAOSS
x1	0.0685	0.0599	0.0518
x2	0.5667	0.573	0.3469
x3	12.204	5.160	14.1675
g1	-0.40	0.0495	-0.1440
g2	-0.341	-0.0243	-0.0281
g3	-1.456	-3.960	-3.2676
g4	-0.576	-0.577	-0.7342

To evaluate the significance of improvements among the proposed algorithms, the Mann–Whitney tests were performed. The results revealed that BAwSSR was significantly better than BADS and BAOSS. However, these two algorithms performed equally; none of them improved significantly over the other. Tables 6.8, 6.9, and 6.10.

Table 6.8 Mann–Whitney significance test at < 0.05 on tension/compression spring for BAwSSR and BADS

	BAwSSR	BADS	p-value
Significance	Yes	NO	0.001

Table 6.9 Mann–Whitney significance test on tension/compression spring for BAOSS and BAwSSR

	BAOSS	BAwSSR	p-value
Significance	No	Yes	0.004

Table 6.10 Mann–Whitney significance test on tension/compression spring for BAOSS and BADS

	BAOSS	BADS	p-value
Significance	NO	Yes	0.420

It can be confirmed from the above analysis that, among all the proposed algorithms, BAwSSR outperformed the others on the tension/compression spring problem. The inference from these findings is that the search space gradual reduction was the most effective improvement among all other methods suggested.

6.3 Summary

In this chapter, the three proposed algorithms were tested on two well-known unconstrained and constrained engineering problems—the gear train problem and the tension/compression spring. To evaluate the performance of these algorithms, the results were compared against a collection of alternatives that were applied successfully to the two engineering problems. Twenty runs were performed to acquire stable figures. Parameters were elicited partially through trial and error and partially from the literature review. In the gear train problem, BAwSSR achieved equal performance with the ABC, PSO–GA, and CS, which was the best among all the algorithms. For the tension/compression spring problem, BAwSSR achieved the best minimum value. However, it could not maintain the same improved performance in all the runs. Nevertheless, the other proposed algorithms, BADS and BAOSS, exhibited the worst performance. It can be assumed that BAwSSR demonstrated improved performance both in the engineering problems and among all three proposed algorithms. This suggests that the search space reduction technique used in BAwSSR proved to be effective. However, its inability to sustain the satisfactory performance in all the testing runs suggests the need for further improvement. Regarding the tension/compression spring, BADS performed remarkably better than all other algorithms involved in the experiment. However, it was not the fastest, even though it needed a slightly higher NFE to achieve its best results.

Chapter 7 CONCLUSION

This chapter summarises the contributions that have been made in this research and offers a conclusion for the whole study. Additionally, it provides suggestions for further enhancements to the methods proposed here.

7.1 Conclusion

To conclude this research, it is worth noting that all objectives stipulated in Chapter 1 have been satisfied. The study aimed at improving the performance of the BA. All proposed enhancements targeted the initialisation and global search stages. The first objective was to enhance the BA using the gradual decrease of the search space to direct the search process to become faster and more focused. The results indicated that this improvement affected the accuracy and speed positively. It also showed an acceptable level of consistent and stable behaviour. The second objective was to use the segmentation of search space into independent samples to improve the BA. The sampling process itself was performed by using one segment to extract one sample with all its different parameters; this is a one-sample-per-segment approach. Another approach was to extract every parameter of the same sample from different segments; this is a one-parameter-per-segment approach. The obtained data showed noticeably consistent behaviour and good improvement in terms of both accuracy and speed.

The third objective was to develop an overlapping segmentation strategy of the search space. This strategy is the opposite of segmenting the search space into independent segments. It allowed the BA to handle different types of problems with one sampling approach, and the outcome had noticeable improvements.

All the previous methods were shown to improve the BA successfully, even in functions with multipocketed topography, except for a few functions, such as Langerman and Bukin6, which always represented a challenge to all algorithms involved in the testing. Some of the proposed enhancements were able find the exact optimum. However, in some functions, tuning of the ngh and n parameters affected the performance positively, which involved a sensitivity issue. Except for BAwSSR, the other two proposed algorithms, BADS and BAOSS, performed poorly on the engineering problems. This might be understood in light of the No Free Lunch Theorem, which states that no one algorithm can exhibit average performance on all classes of problems (Wolpert and Macready, 1997).

7.2 Contribution

This research examined possibilities for improving the BA by manipulating the search space with different techniques focusing on the global search and initialisation stages. The work explored the effects of improving these two stages on the overall performance of the BA.

- The research contributed to proving the ability to improve the overall performance of the BA in terms of accuracy via improving the initialisation and global search stages. The proposed methods showed positive effects in reducing the number of function evaluations (NFEs) and improving the success rate (SR), which accelerated the search for the optimum.
- One of the research contributions was to improve the accuracy of the BA with the ability to find the exact minimum in some benchmark problems. A new search method was introduced based on continuous and gradual decreases in search space with different scenarios to better utilise the generated samples. The additional results for two engineering problems indicated enhanced performance.

- Another contribution was to introduce two sampling techniques:
    1. To take all the parameters of one sample from the same segments.
    2. To take every parameter of one sample from different segments.

  The effectiveness of this method is due to its flexibility in handling different types of problems. The testing of this method indicated considerable enhancements in performance.
- A new method of segmentation was introduced based on overlapping segmentation of the search space. The purpose was to permit flexible handling of different types of problems while using the same sampling technique. The testing of this method indicated positive effects on performance.

7.3 Future work

One of the methods used in this research was based on the gradual decrease of search space in five scenarios. Expansions rather than a decrease of search space could have also been used.

Additionally, the selection of these scenarios was not based on a guaranty of their optimality; rather, it was based on trial and error. Logically, many other scenarios could have been used and need further investigation. Some of the methods formulated in this research used a fixed segmentation approach; however, a variable segmentation approach could also be used.

While good results were obtained from targeting only the initialisation and global search stages, they could also be extended to apply to neighbourhood searches. Another critical issue in this research is the poor performance of BADS and BAOSS in engineering problems, which contradicts the good performance on the benchmark function. There is a need to investigate this weakness and to further improve the method used. While BAwSSR was able to obtain a high value among the best values found, the means and standard deviations indicate that it could not sustain the good performance in all the runs; this implies the potential for further enhancement. Moreover, during the testing of the engineering problems, the handling of parameters of different domains of values, as well as inequality constraints, represented a challenge in the search for the optimum. Better handling of the constraints would affect a decrease in the processing time and accuracy.

During the testing of the suggested techniques, it was noted that they are sensitive to the ngh and n parameters. An exploration of this issue will likely help the proposed algorithms to become more robust and reliable. None of the proposed algorithms were applied on combinatorial domain problems, and it is worth investigating the effects on segmentation on combinatorial problems. However, major modifications to these algorithms might be required to handle these types of problems. Furthermore, the field of multi-objective optimisation has not been studied in this research and it is worth investigating how to apply the developed algorithms on multi-objective problems.

REFERENCES

Abidin, Z. Z., Arshad, M. R. and Ngah, U. K., 2011. A Simulation Based Fly Optimization Algorithm for Swarms of Mini Autonomous Surface Vehicles Application. *indian Journal of Geo-Marine Sciences,* 40(2), pp.250-266,

Ahmad, S. A., Pham, D. T., Ng, K. W. and Ang, M. C., 2012. TRIZ-inspired Asymmetrical Search Neighborhood in the Bees Algorithm. In *IEEE 2012 Sixth Asia Modelling Symposium (AMS), Bali 2012*, pp.29-33. doi: 10.1109/AMS.2012.30.

Akay, B. and Karaboga D. 2012. Artificial bee colony algorithm for large-scale problems and engineering design optimization. *Journal of intelligent manufacturing*, *23*(4), pp.1001–1014, https://doi.org/10.1007/s10845-010-0393-4

Alfi, A. and Khosravi, A., 2012. Constrained Nonlinear Optimal Control via a Hybrid BA-SD. *International Journal of Engineering-Transactions C: Aspects* [Online], 25(3): 197-204. Available from http://www.ije.ir [Accessed 10 July 2020].

Aljahdali, S.H, Ghiduk, A.S and El-Telbany, M. 2010. The limitations of genetic algorithms in software testing. In *ACS/IEEE International Conference on Computer Systems and Applications-AICCSA, Hammamet 2010,* pp. 1–7, doi: 10.1109/AICCSA.2010.5586984.

Anantasate, S., Chokpanyasuwan, C. and Bhasaputra, P., 2010. Optimal power flow by using bees algorithm. In *ECTI-CON2010: The 2010 ECTI International Confernce on Electrical Engineering/Electronics, Computer, Telecommunications and Information Technology,* Chiang Mai, Thailand, pp. 430-434.

Anderson,C. and Ratnieks, F. 1999. Worker allocation in insect societies: coordination of nectar foragers and nectar receivers in honeybee (Apis mellifera) colonies. *Behavioral Ecology and Sociobiology*, *46*(2), pp.73–81, https://doi.org/10.1007/s002650050595

Azfanizam, A.S., Pham, D.T., Faieza, A.A., 2014. Combination of Adaptive Enlargement and Reduction in the Search Neighbourhood in the Bees Algorithm. AMM 564, pp.614–618, https://doi.org/10.4028/www.scientific.net/amm.564614 [Accessed 12 July 2020].

Baghel, M, Agrawal S, Silakari S, 2012. Survey of metaheuristic algorithms for combinatorial optimization. *International Journal of Computer Applications, 58*(19), pp. 21-31.

Battiti, R. and Brunato, M., 2010. Reactive search optimization: learning while optimizing. In *Handbook of Metaheuristics* (pp. 543-571). Springer, Boston, MA, https://doi.org/10.1007/978-1-4419-1665-5_18

Beheshti, Z. and Shamsuddin, S.M.H., 2013. A review of population-based meta-heuristic algorithms. *Int. J. Adv. Soft Comput. Appl*, *5*(1), pp.1-35.

Bhattacharjya, R.K. 2009. CE 602: Optimization Method, Region Elimination Method, Indian Institute of Technology, Guwahati.

Biswal, B., Dash, P.K. and Mishra, S., 2011. A hybrid ant colony optimization technique for power signal pattern classification. *Expert Systems with Applications*, *38*(5), pp.6368-6375, https://doi.org/10.1016/j.eswa.2010.11.102.

Blondin, J., 2009. Particle swarm optimization: A tutorial. Availaible from site: http://cs.armstrong.edu/saad/csci8100/pso_tutorial.Pdf , [Accessed 15 July 2020].

Bonabeau, E., Dorigo, M. and Theraulaz, G., 1999. From Natural to Artificial Swarm Intelligence. Oxford University Press, Inc., USA.

Boussaïd, I., Lepagnot, J. and Siarry, P., 2013. A survey on optimization metaheuristics. *Information sciences*, *237*, pp.82–117, https://doi.org/10.1016/j.ins.2013.02.041.

Chaparro, I. and Valdez, F., 2013. Variants of ant colony optimization: a metaheuristic for solving the traveling salesman problem. *Recent Advances on Hybrid Intelligent Systems*, Springer, Berlin, 451,pp. 323-331, https://doi.org/10.1007/978-3-642-33021-6_26.

Chinneck, J.W., 2006. *Practical optimization: a gentle introduction*. Systems and Computer Engineering), Carleton University, Ottawa, Canada [online] Available *http://www. sce. carleton. ca/faculty/chinneck/po. html*, p.11. [Accessed 14 july 2020]

Dahiya, S.S., Chhabra, J.K. and Kumar, S., 2010. Application of artificial bee colony algorithm to software testing. In *2010 21st Australian software engineering conference*, Auckland, pp. 149-154, doi: 10.1109/ASWEC.2010.30.

Deb, K., 2012. *Optimization for engineering design: Algorithms and examples*. PHI Learning Pvt. Ltd., New Delhi, India.

Derakhshan, M. and Shirazi, K.H., 2014. Optimized fuzzy controller for a power–torque distribution in a hybrid vehicle with a parallel configuration. *Proceedings of the Institution of Mechanical Engineers, Part D: Journal of Automobile Engineering*, *228*(14), pp.1654-1674, https://doi.org/10.1177/0954407013496183.

Dieterich, J.M. and Hartke, B., 2012. Empirical Review of Standard Benchmark Functions Using Evolutionary Global Optimization. *Applied Mathematics*, *3*, pp.1552-1564

Ding, Q., Hu, X., Sun, L. and Wang, Y., 2012. An improved ant colony optimization and its application to vehicle routing problem with time windows. *Neurocomputing*, 98, pp.101-107.

Dorigo, M., Maniezzo, V. and Colorni, A.,1991. *Positive feedback as a search strategy*. Technical Report 91-016, viewed 14th August 2020, Dipartimento di Elettronica, Politecnico di Milano, Milan, Italy, https://doi.org/10.1016/j.neucom.2011.09.040

Dorigo, M., Maniezzo, V. and Colorni, A., 1996. Ant system: optimization by a colony of cooperating agents. *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, 26(1), pp.29–41, doi: 10.1109/3477.484436.

Dorigo, M. and Di Caro, G., 1999. Ant colony optimization: a new meta-heuristic. In *Proceedings of the 1999 congress on evolutionary computation-CEC99 (Cat. No. 99TH8406)* , IEEE, Washington, DC, USA, 2, pp. 1470–147, doi:10.1109/CEC.1999.782657.

Dorigo, M., Birattari, M. and Stutzle, T., 2006. Ant colony optimization. *IEEE computational intelligence magazine*, 1(4), pp.28–39, doi: 10.1109/MCI.2006.329691.

Dos Santos Coelho, L. and Alotto, P., 2011. Gaussian artificial bee colony algorithm approach applied to Loney's solenoid benchmark problem. *IEEE Transactions on Magnetics*, *47*(5), pp.1326-1329.

Eldukhri, E.E. and Kamil, H.G., 2015. Optimisation of swing-up control parameters for a robot gymnast using the Bees Algorithm. *Journal of Intelligent Manufacturing*, *26*(5), pp.1039-1047, doi: 10.1109/TMAG.2010.2087317.

Eswaramurthy, V.P. and Tamilarasi, A., 2009. Hybridizing tabu search with ant colony optimization for solving job shop scheduling problems. *The International Journal of Advanced Manufacturing Technology*, *40*(9–10), pp.1004–1015, DOI: 10.1007/s00170-008-1404-x

Gandelli, A., Grimaccia, F., Mussetta, M., Pirinoli, P. and Zich, R.E., 2007. Development and validation of different hybridization strategies between GA and PSO. In *2007 IEEE Congress on Evolutionary Computation*, pp. 2782–2787, doi: 10.1109/CEC.2007.4424823.

Gandomi, A.H., Yang, X.S. and Alavi, A.H., 2013. Cuckoo search algorithm: a metaheuristic approach to solve structural optimization problems. *Engineering with computers*, 29(1), pp.17–35, DOI 10.1007/s00366-011-0241-y

Garg, H., 2016. A hybrid PSO-GA algorithm for constrained optimization problems. *Applied Mathematics and Computation*, 274, pp.292–305, https://doi.org/10.1016/j.amc.2015.11.001

Glover, F., 1990. Tabu search: A tutorial. *Interfaces*, *20*(4), pp.74-94, https://doi.org/10.1287/inte.20.4.74

Glover, F.W. and Kochenberger, G.A. eds., 2006. *Handbook of metaheuristics*, *International Series in Operations Research & Management Science,*57, Springer Science & Business Media, Boston.

Gordon, D.M., 1996. The organization of work in social insect colonies. *Nature*, *380*(6570), pp.121–124.

Guo, C.X., Hu, J.S., Ye, B. and Cao, Y.J., 2004. Swarm intelligence for mixed-variable design optimization. *Journal of Zhejiang University-SCIENCE A*, *5*(7), pp.851–860, https://doi.org/10.1631/jzus.2004.0851.

Hansen, P., Mladenović, N. and Pérez, J.A.M., 2008. Variable neighbourhood search: methods and applications. *Operational Research*, *6*(4), pp.319–360, https://doi.org/10.1007/s10479-009-0657-6.

Hansen N., Finck S., Raymond Ros R., Anne Auger A., 2009. Real-Parameter Black-Box Optimization Benchmarking 2009: Noiseless Functions Definitions. [Research Report] RR-6829, 2009. <inria-00362633>

He, S., Prempain, E. and Wu, Q.H., 2004. An improved particle swarm optimizer for mechanical design optimization problems. *Engineering optimization*, *36*(5), pp.585–605.

Herrera, F., Lozano, M. and Verdegay, J.L., 1998. Tackling real-coded genetic algorithms: Operators and tools for behavioural analysis. *Artificial intelligence review*, 12(4), pp.265–319, https://doi.org/10.1023/A:1006504901164.

Hu, X., Ding, Q. and Wang, Y., 2010. A hybrid ant colony optimization and its application to vehicle routing problem with time windows. In *Life System Modelling and Intelligent Computing,*Wuxi, China, Springer, Berlin, Germany, pp. 70-76, https://doi.org/10.1007/978-3-642-15853-7_10

Huang, S.J. and Liu, X.Z., 2013. Application of artificial bee colony-based optimization for fault section estimation in power systems. *International Journal of Electrical Power & Energy Systems*, *44*(1), pp.210-218, https://doi.org/10.1016/j.ijepes.2012.07.012.

Hussein, W.A., Sahran, S. and Abdullah, S.N.H.S., 2014. Patch-Levy-based initialization algorithm for Bees Algorithm. *Applied Soft Computing*, 23, pp.104–121, https://doi.org/10.1016/j.asoc.2014.06.004.

Hyma, J., Jhansi, Y. and Anuradha, S., 2010. A new hybridized approach of PSO & GA for ‘

document clustering. *International Journal of Engineering Science and Technology*, 2(5), pp.1221–1226.

Imanguliyev, A., 2013. *Enhancements for the Bees Algorithm* (Doctoral dissertation, Cardiff University).

Jamil, M. And Yang, X., 2013. A Literature Survey of Benchmark Functions for Global Optimisation Problems. *Journal of Numerical Optimisation*, 4(2), pp. 150-194, https://doi.org/10.1504/IJMMNO.2013.055204

Kamsani, S.H., 2016. *Improvements on the bees algorithm for continuous optimisation problems* (Doctoral dissertation, University of Birmingham).

Kannan, B. K. and Kramer, S. N. 1994. Augmented Lagrange Multiplier Based Method for Mixed Integer Discrete Continuous Optimization and Its Applications to Mechanical Design. *Journal of Mechanical Design, Transactions of the ASME* 116(2), pp. 405-411, https://doi.org/10.1115/1.2919393

Karaboga, D., 2005. *An idea based on honeybee swarm for numerical optimization*, pp. 1-10, Technical report-tr06, Erciyes university, engineering faculty, computer engineering department.

Karaboga, D. and Basturk, B., 2007. A powerful and efficient algorithm for numerical function optimization: artificial bee colony (ABC) algorithm. *Journal of global optimization*, *39*(3), pp.459-471, DOI 10.1007/s10898-007-9149-x

Karaboga, D. and Gorkemli, B., 2012. A quick artificial bee colony-qABC-algorithm for optimization problems. *2012 International symposium on innovations in intelligent systems and applications*, IEEE, Trabzon, Turkey, pp. 1-5, https://doi.org/10.1023/A:1006504901164

Kennedy, J. and Eberhart, R., 1995. Particle swarm optimization. In *Proceedings of ICNN'95-International Conference on Neural Networks*, IEEE, Perth, WA, Australia, 4, pp. 1942–1948, doi: 10.1109/ICNN.1995.488968.

Kennedy, J. and Eberhart, R.C., 1997. A Discrete Binary Version of the Particle Swarm Algorithm. *1997 IEEE International conference on systems, man, and cybernetics. Computational cybernetics and simulation*, IEEE, Orlando, FL, USA 5, pp. 4104–4108, doi: 10.1109/ICSMC.1997.637339.

Kıran, M.S. and Gündüz, M., 2013. A recombination-based hybridization of particle swarm optimization and artificial bee colony algorithm for continuous optimization problems. *Applied Soft Computing*, *13*(4), pp.2188-2203, https://doi.org/10.1016/j.asoc.2012.12.007.

Kirkpatrick, S., Gelatt, C.D. and Vecchi, M.P., 1983. Optimization by simulated annealing. *science*, 220(4598), pp.671–680, DOI: 10.1126/science.220.4598.671

Korb, O., Stützle, T. and Exner, T.E., 2007. An ant colony optimization approach to flexible protein–ligand docking. *Swarm Intelligence, 1*(2), pp.115–134, https://doi.org/10.1007/s11721-007-0006-9

Kumar, A. and Kumar, V., 2017. Hybridized ABC-GA optimized fractional order fuzzy pre- compensated FOPID Control design for 2-DOF robot manipulator. *AEU-International Journal of Electronics and Communications*, *79*, pp. 219–233, https://doi.org/10.1016/j.aeue.2017.06.008.

Laskari, E.C., Parsopoulos, K.E. and Vrahatis, M.N., 2002. Particle swarm optimization for integer programming. In *Proceedings of the 2002 Congress on Evolutionary Computation. CEC'02 (Cat. No. 02TH8600),*IEEE, Honolulu, HI, USA, 2, pp. 1582–1587.

Liang, J.J., Qin, A.K., Suganthan, P.N. and Baskar, S., 2006. Comprehensive learning particle swarm optimizer for global optimization of multimodal functions. *IEEE transactions on evolutionary computation*, *10*(3), pp.281–295, doi: 10.1109/TEVC.2005.857610.

Lourenço, H.R., Martin, O.C. and Stützle, T., 2003. Iterated local search, In: Glover F., Kochenberger G.A. (eds)*Handbook of metaheuristics,* Springer, Boston, MA, 57. pp. 320–353, https://doi.org/10.1007/0-306-48056-5_11

Luke, S., 2013. *Essentials of metaheuristics*,Lulu. available online at http://cs.gmu.edu/∼sean/book/metaheuristics/ [Accessed on 11 July 2020]

Mei, C.A., Pham, D.T., Anthony, J.S. and Kok, W.N., 2010. PCB assembly optimisation using the Bees Algorithm enhanced with TRIZ operators. *IECON 2010-36th Annual Conference on IEEE Industrial Electronics Society,* Glendale, AZ, USA, pp. 2708-2713, doi: 10.1109/IECON.2010.5675114.

Mezura-Montes, E. and Coello, C. A. 2005. Useful infeasible solutions in engineering optimization with evolutionary algorithms. MICAI 2005: Advances in Artificial Intelligence. MICAI 2005, Springer, Berlin, Germany, pp. 652-662, https://doi.org/10.1007/11579427_66

Mladenović, N. and Hansen, P., 1997. Variable neighborhood search. *Computers & operations research*, *24*(11), pp.1097-1100, https://doi.org/10.1016/S0305-0548(97)00031-2.

Murata, T. and Ishibuchi, H., 1994. Performance evaluation of genetic algorithms for flowshop scheduling problems. In *Proceedings of the First IEEE Conference on Evolutionary Computation. IEEE World Congress on Computational Intelligence, , Orlando, FL, USA*, 2, pp. 812–817, doi: 10.1109/ICEC.1994.349951.

Nachar, N., 2008. The Mann-Whitney U: A test for assessing whether two independent samples come from the same distribution. *Tutorials in Quantitative Methods for Psychology*, *4*(1), pp.13–20.

Nievergelt, J., Gasser, R., Mäser, F. and Wirth, C., 1995. All the needles in a haystack: Can exhaustive search overcome combinatorial chaos? In *Computer Science Today*, 1000, pp. 254-274, https://doi.org/10.1007/BFb0015248.

Niu, D., Gu, Z. and Xing, M., 2007. Research on neural networks based on culture particle swarm optimization and its application in power load forecasting. In *Third International Conference on Natural Computation (ICNC 2007), Haikou, China*, IEEE, 1, pp.270–274, doi: 10.1109/ICNC.2007.627.

Nolle, L., Köppen, M., Schaefer, G. and Abraham, A., 2011. Intelligent computational optimization in engineering: Techniques and applications. *Studies in Computational Intelligence*, *Springer*, 366, pp. 1-24, https://doi.org/10.1007/978-3-642-21705-0_1.

Owen, C.B. and Abunawass, A.M., 1993. Application of simulated annealing to the backpropagation model improves convergence. In *Proceedings of the SPIE Conference on Science of Artificial Neural Networks II, Orlando, FL, USA,1966*, SPIE, pp. 269–276, https://doi.org/10.1117/12.152626.

Omran, M.G., Engelbrecht, A.P. and Salman, A., 2004. Image classification using particle swarm optimization. In *Recent advances in simulated evolution and learning*, pp. 347–365.

Onwunalu, J.E. and Durlofsky, L.J., 2010. Application of a particle swarm optimization algorithm for determining optimum well location and type. *Computational Geosciences*, *14*(1), pp.183–198, DOI 10.1007/s10596-009-9142-1

Packianather, M.S., Yuce, B., Mastrocinque, E., Fruggiero, F., Pham, D.T. and Lambiase, A., 2014. Novel Genetic Bees Algorithm applied to single machine scheduling problem. In *2014 World Automation Congress (WAC), Waikoloa, HI, USA*, IEEE, pp. 906-911, doi: 10.1109/WAC.2014.6936194.

Pan, Q.K., Tasgetiren, M.F., Suganthan, P.N. and Chua, T.J., 2011. A discrete artificial bee colony algorithm for the lot-streaming flow shop scheduling problem. *Information sciences*, *181*(12), pp.2455-2468, https://doi.org/10.1016/j.ins.2009.12.025.

Parsopoulos, K.E. and Vrahatis, M.N., 2005. Unified particle swarm optimization for solving constrained engineering optimization problems. *First International conference on natural computation, ICNC 2005, Changsha, China*, Springer, Berlin, Germany, pp. 582–591, https://doi.org/10.1007/11539902_71

Pham, D.T. and Castellani, M., 2015. A comparative study of the Bees Algorithm as a tool for function optimisation. *Cogent Engineering*, *2*(1), pp.1-28, https://doi.org/10.1080/23311916.2015.1091540

Pham, D.T., Castellani, M. and Fahmy, A.A., 2008b. Learning the inverse kinematics of a robot manipulator using the bees algorithm. In *2008 6th IEEE International Conference on Industrial Informatics,* IEEE, pp. 493-498, doi: 10.1109/INDIN.2008.4618151.

Pham, D.T. and Darwish, H.A., 2010. Using the bees algorithm with Kalman filtering to train an artificial neural network for pattern classification. *Proceedings of the Institution of Mechanical Engineers, Part I: Journal of Systems and Control Engineering*, *224*(7), pp.885–892, https://doi.org/10.1243/09596518JSCE1004

Pham, D.T., Haj Darwish, A. and Eldukhri, E.E., 2009. Optimisation of a fuzzy logic controller using the bees algorithm. *International Journal of Computer Aided Engineering and Technology*, *1*(2), pp.250-264, https://doi.org/10.1504/IJCAET.2009.02279

Pham, D.T., Ghanbarzadeh, A., Koc, E., Otri, S., Rahim, S. and Zaidi, M., 2005. The bees algorithm Technical Note, *Manufacturing Engineering Centre, Cardiff University, UK*.

Pham, D.T., Koc, E., Lee, J.Y. and Phrueksanant, J., 2007. Using the bees algorithm to schedule jobs for a machine. In *Proceedings Eighth International Conference on Laser Metrology, CMM and Machine Tool Performance, LAMDAMAP, Euspen, UK, Cardiff*, pp. 430-439.

Pham, D.T., Pham, Q.T., Ghanbarzadeh, A. and Castellani, M., 2008a. Dynamic optimisation of chemical engineering processes using the bees algorithm. *IFAC Proceedings Volumes*, *41*(2), pp.6100-6105, https://doi.org/10.3182/20080706-5-KR-1001.01030.

Prakasam, A. and Savarimuthu, N., 2016. Metaheuristic algorithms and probabilistic behaviour: a comprehensive analysis of Ant Colony Optimization and its variants. *Artificial Intelligence Review*, *45*(1), pp.97–130, DOI 10.1007/s10462-015-9441-y

Qi, X. and Palmieri, F., 1994. Theoretical analysis of evolutionary algorithms with an infinite population size in continuous space. Part II: Analysis of the diversification role of crossover. *IEEE Transactions on Neural Networks*, *5*(1), pp.120–129, doi:10.1109/72.265965

Qiu, J., Yang, D. and Xie, J., 2013. An artificial bee colony algorithm with modified search strategies for global numerical optimization, *Journal of Theoretical and Applied Information Technology*, 48(1) pp.1-10

Ray, T. and Liew, K. M. 2003. Society and civilization: An optimization algorithm based on the simulation of social behavior. *IEEE Transactions on Evolutionary Computation* 7(4), pp. 386-396, doi: 10.1109/TEVC.2003.814902.

Resende, M.G., Martí, R. and Panos, P., 2018. *Handbook of Heuristics*, Springer, Cham, Switzerland

Rosenbrock, H., 1960. An automatic method for finding the greatest or least value of a function. *The Computer Journal*, *3*(3), pp.175–184, https://doi.org/10.1093/comjnl/3.3.175

Satheesh, A. and Manigandan, T. Maintaining Power System Stability with Facts Controller using Bees Algorithm and NN, *Journal of Theoretical and Applied Information Technology*, 49, (1), pp. 38-47

Savsani, P., Jhala, R.L. and Savsani, V., 2014. Effect of hybridizing biogeography-based optimization (BBO) technique with artificial immune algorithm (AIA) and ant colony optimization (ACO). *Applied Soft Computing*, 21, pp. 542–553, https://doi.org/10.1016/j.asoc.2014.03.011.

Sun, P., Zhang, Y., Liu, J. and Bi, J., 2020 "An Improved Atom Search Optimization With Cellular Automata, a Lévy Flight and an Adaptive Weight Strategy," in *IEEE Access*, 8, pp. 49137-49159, doi: 10.1109/ACCESS.2020.2979921.

Seeley, T.D., 2002. When is self-organization used in biological systems? *The Biological Bulletin*, *202*(3), pp. 314–318.

Shafia, M.A., Moghaddam, M.R., & Tavakolian, R.,2011. A Hybrid Algorithm for Data Clustering Using Honey Bee Algorithm, Genetic Algorithm and K-Means Method. *Journal of Advanced Computer Science and Technology, 1*, pp.110-125.

Shanghooshabad, A.M. and Abadeh, M.S., 2016. Robust, interpretable, and high quality fuzzy rule discovery using krill herd algorithm. *Journal of Intelligent & Fuzzy Systems*, *30*(3), pp.1601–1612.

Shatnawi, N., Sahran, S. and Faidzul, M., 2013. A memory-based Bees Algorithm: an enhancement. *JApSc*, *13*(3), pp.497-502, DOI: 10.3923/jas.2013.497.502.

Shtovba, S.D., 2005. Ant algorithms: theory and applications. *Programming and Computer Software*, *31*(4), pp.167–178, https://doi.org/10.1007/s11086-005-0029-1

Sorensen, K., Sevaux, M. and Glover, F., 2017. A history of metaheuristics. *arXiv preprint arXiv:1704.00853*.

Thammano, A. and Phu-ang, A., 2013. A hybrid artificial bee colony algorithm with local search for flexible job-shop scheduling problem. *Procedia computer science, 20,* pp.96-101, https://doi.org/10.1016/j.procs.2013.09.245.

Van den Bergh, F. and Engelbrecht, A.P., 2002. A new locally convergent particle swarm optimiser. In *IEEE International conference on systems, man, and cybernetics,* Yasmine Hammamet, Tunisia, IEEE,3, pp. 1-6, doi: 10.1109/ICSMC.2002.1176018.

Wang, S., Zhang, Y., Dong, Z., Du, S., Ji, G., Yan, J., Yang, J., Wang, Q., Feng, C. and Phillips, P., 2015. Feed-forward neural network optimized by hybridization of PSO and ABC for abnormal brain detection. *International Journal of Imaging Systems and Technology, 25*(2), pp.153–164, https://doi.org/10.1002/ima.22132.

Weise, T., 2009. *Global optimization algorithms-theory and application*. Self-Published Thomas Weise.

Whitley, D., 1994. A genetic algorithm tutorial. *Statistics and computing*, *4*(2), pp.65–85, https://doi.org/10.1007/BF00175354

Wolpert, D.H. and Macready, W.G., 1997. No free lunch theorems for optimization. *IEEE transactions on evolutionary computation*, *1*(1), pp.67–82, doi: 10.1109/4235.585893.

Yang, F., Li, Z. and Fan, Y., 2015. A Specific Combination Scheme for Communication Modulation Recognition Based on the Bees Algorithm. In *2015 International Conference on Mechatronics, Electronic, Industrial and Control Engineering (MEIC-15),* Atlantis Press, pp.185-188, https://doi.org/10.2991/meic-15.2015.45

Yang, X.S., Deb, S., Zhao, Y.X., Fong, S. and He, X., 2018. Swarm intelligence: past, present and future. *Soft Computing*, *22*(18), pp.5923-5933, https://doi.org/10.1007/s00500-017-2810-5

Yuce, B., Fruggiero, F., Packianather, M.S., Pham, D.T., Mastrocinque, E., Lambiase, A. and Fera, M., 2017. Hybrid Genetic Bees Algorithm applied to single machine scheduling with earliness and tardiness penalties. *Computers & Industrial Engineering*, *113*, pp.842-858, https://doi.org/10.1016/j.cie.2017.07.018

Yuce, B., Pham, D.T., Packianather, M.S. and Mastrocinque, E., 2015. An enhancement to the Bees Algorithm with slope angle computation and Hill Climbing Algorithm and its applications on scheduling and continuous-type optimisation problem. *Production & Manufacturing Research*, *3*(1), pp.3–19, https://doi.org/10.1080/21693277.2014.976321

Zambrano-Bigiarini, M., Clerc, M. and Rojas, R., 2013. Standard particle swarm optimisation 2011 at cec-2013: A baseline for future pso improvements. In *2013 IEEE Congress on Evolutionary Computation* (pp. 2337-2344). IEEE, DOI: 10.1109/CEC.2013.6557848

Zabil, M.H.M. and Zamli, K.Z., 2013. Implementing a T-way test generation strategy using Bees Algorithm. *International Journal of. Advances in Soft Computing and its Applications, 5,* (3) [, pp.116-126.

Zhu, G. and Kwong, S., 2010. Gbest-guided artificial bee colony algorithm for numerical function optimization. *Applied mathematics and computation*, *217*(7), pp.3166-3173, https://doi.org/10.1016/j.amc.2010.08.049

Zukhri, Z. and Paputungan, I.V., 2013. A hybrid optimization algorithm based on genetic algorithm and ant colony optimization. *International Journal of Artificial Intelligence & Applications*, *4*(5), p.1-13, DOI : 10.5121/ijaia.2013.4505

APPENDICES

Appendix A GEAR TRAIN DESIGN PROBLEM

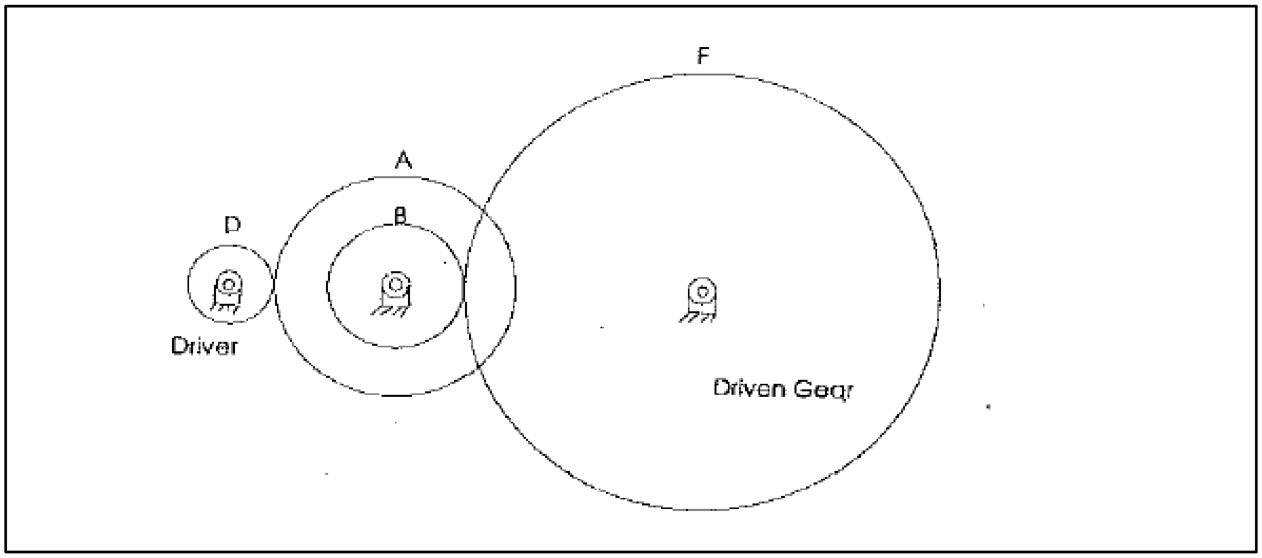


Figure A.1 Gear train design scheme

Figure A.1 represents the gear train design problem scheme. The gear train consists of two sets of gear wheels: a–d and b–f. The formula for the gear ratio is as follows:

gear ratio = $\frac{T_d T_b}{T_a T_f}$.

This ratio is required to be close enough to the value 1/6.931. The objective functions of this problem can be formulated as:

Min $f(x) = \left(\frac{1}{6.931} - \frac{x_3 x_2}{x_1 x_4}\right)^2$ Subject to $12 \leq x_i \leq 60$ i $= 1,2,3,4$

where x1, x2, x3, and x4 are the numbers of teeth in the gears d, b, a, and f and they take only integer values.

Appendix B COMPRESSION TENSION SPRING

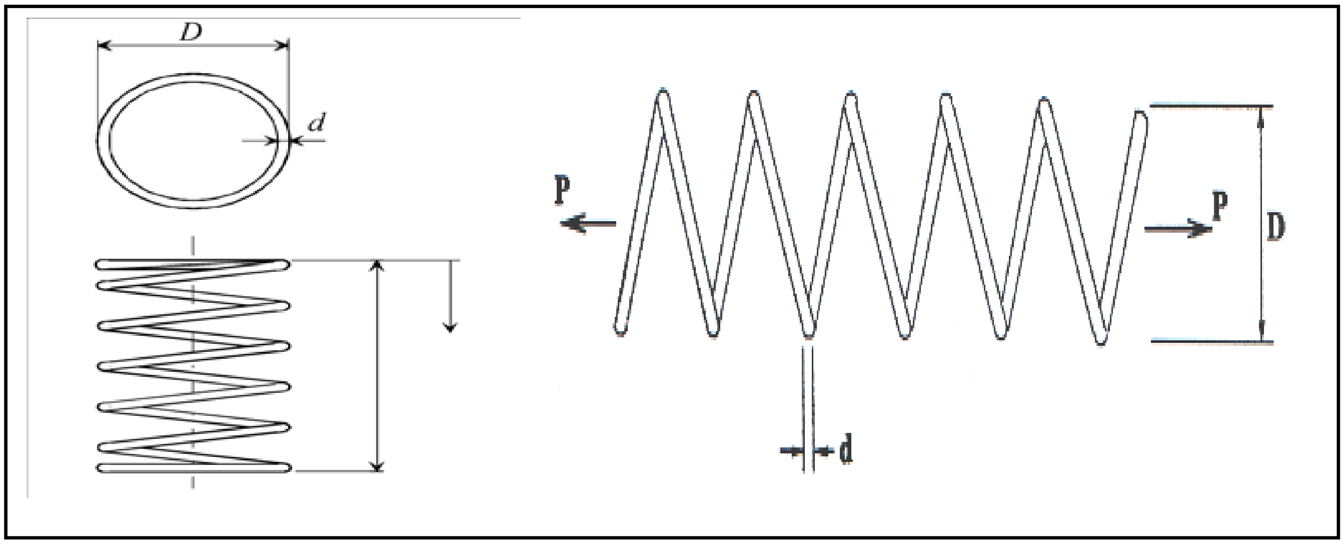


Figure B.1 Compression/tension spring

Assuming that $d = x_1\,, D = x_2\,, P = x_3$, the formula of the compression/tension spring is

Min f (x) = $(\boldsymbol{x_3} + \mathbf{2})\boldsymbol{x_2 x_1^2}$

subject to the following constraints:

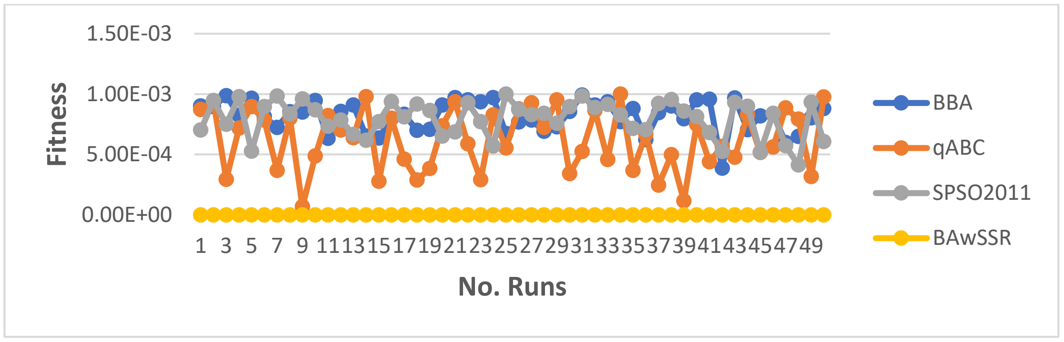
$$g_1(x) = 1 - \frac{x_2^3 x_3}{71785 x_1^4} \leq 0$$

$$g_2(x) = \frac{4x_2^2 - \boldsymbol{x_1 x_2}}{12566(x_2\, x_1^3 - x_1^4)} + \frac{1}{5108 x_1^2} - 1 \leq 0$$

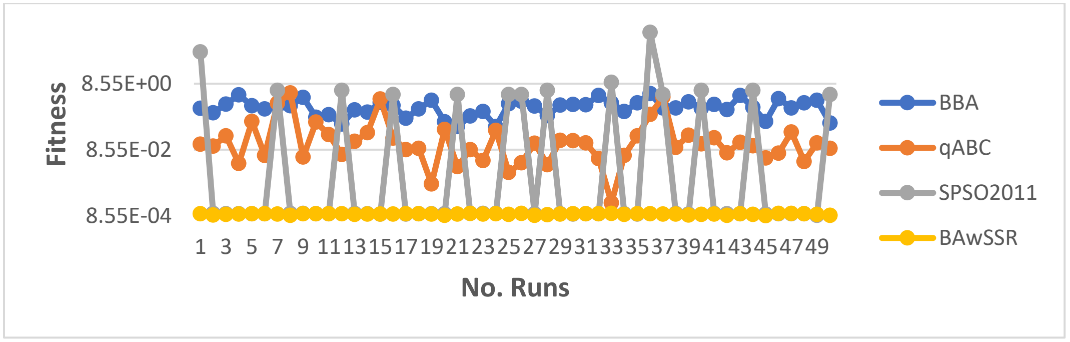
$$g\_3\ (x) = 1 - (140.45x\_1)/(x\_2\hat{}2\ x\_3\ ) \leq 0$$

$$\boldsymbol{g\_4\ (x) = (x\_2 + x\_1)/(1.5)\ - 1\ \leq 0}.$$

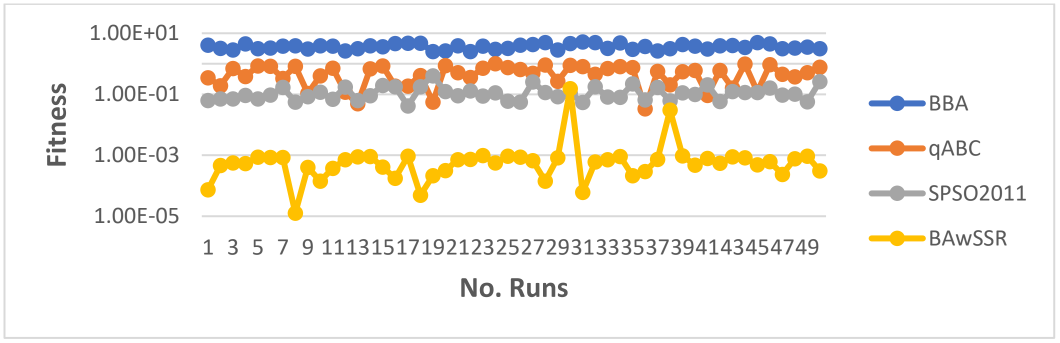
Appendix C LIST OF CHARTS FOR BAWSSR ALGORITHM



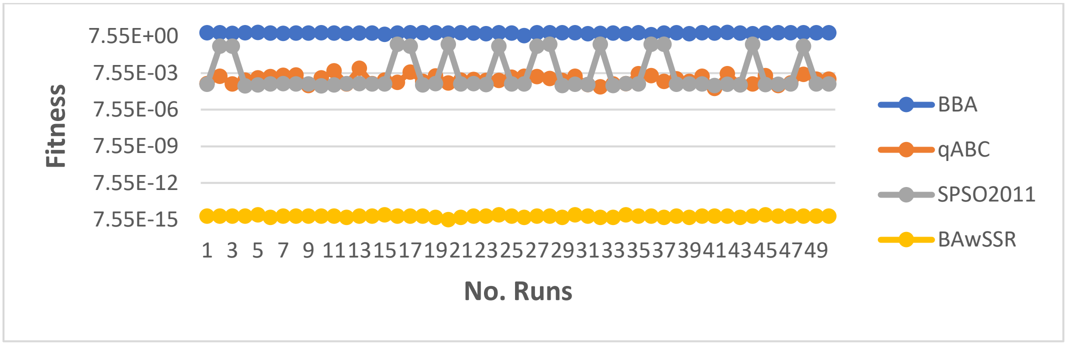
Accuracy figures of 50 runs- Sphere function



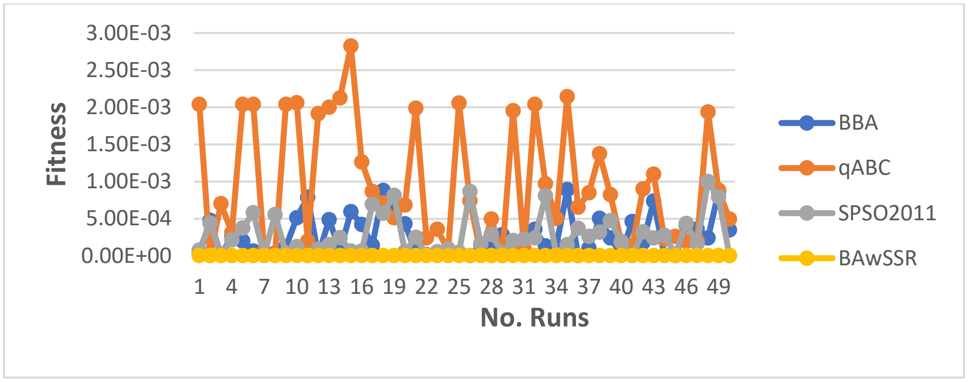
Accuracy figures for 50 runs-Rosenbrock function



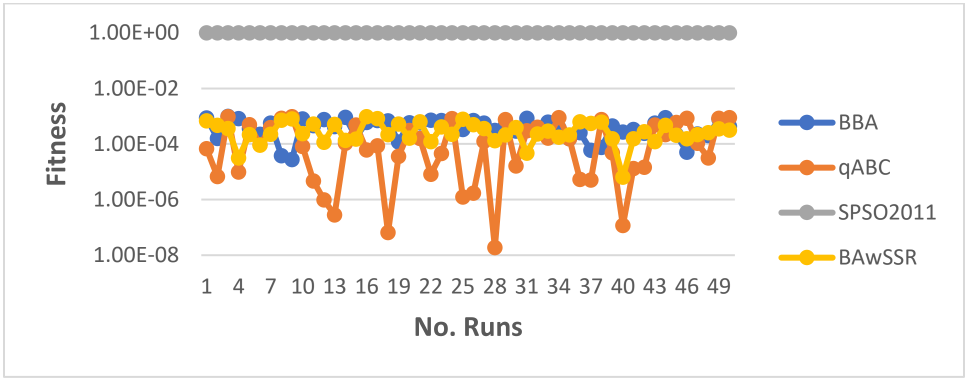
Accuracy figures for 50 runs- Quartic function



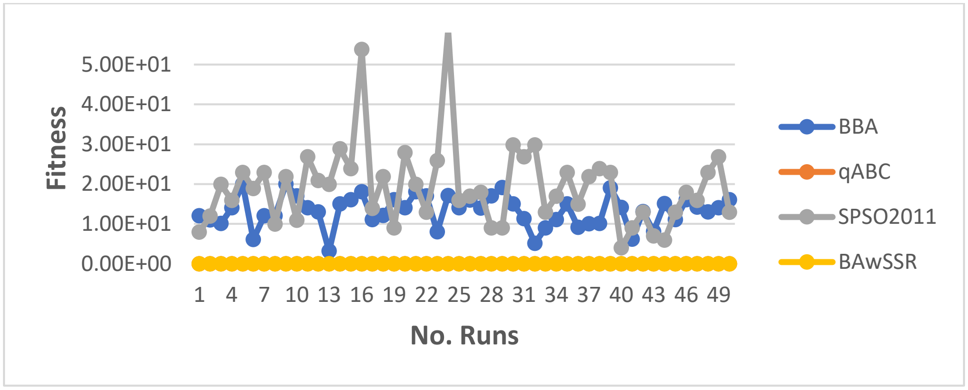
Accuracy figures for 50 runs- Ackley function



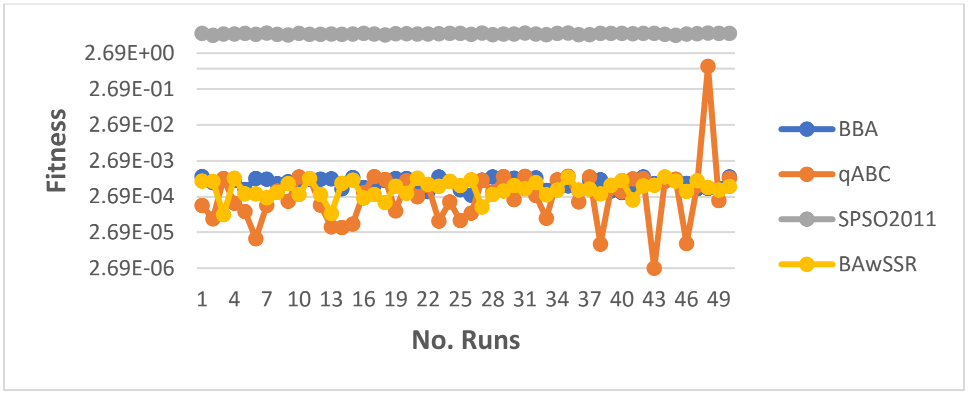
Accuracy figures for 50 runs- Schaffer function



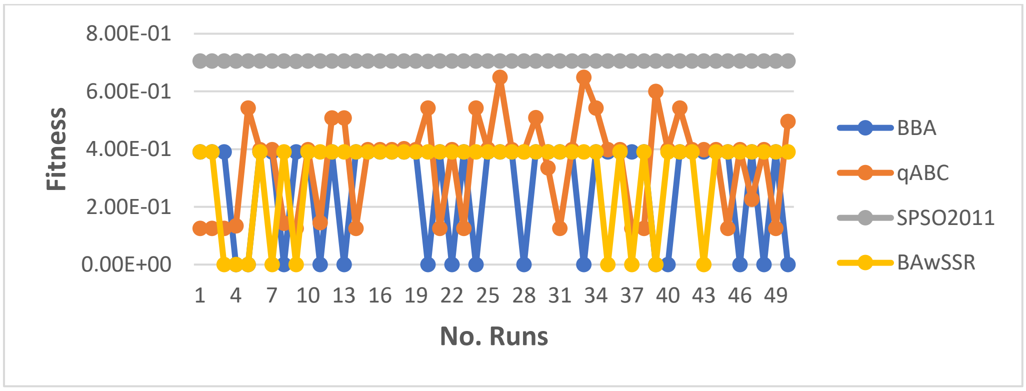
Accuracy figures for 50 runs- Easom function



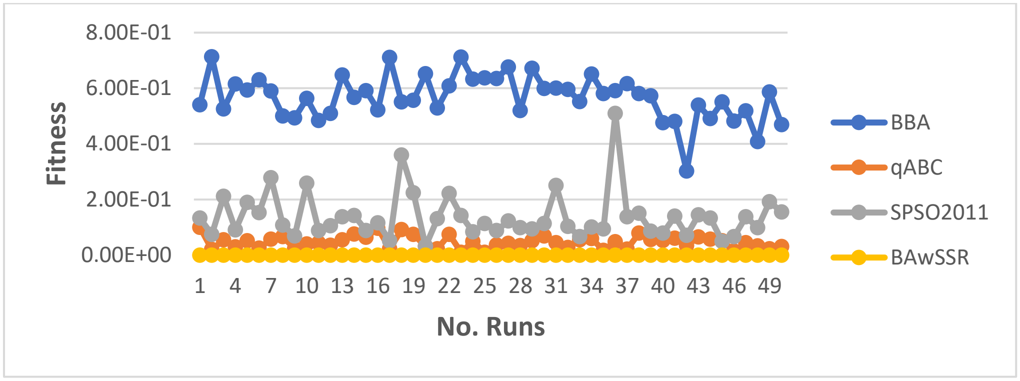
Accuracy figures for 50 runs-Rastrigin function



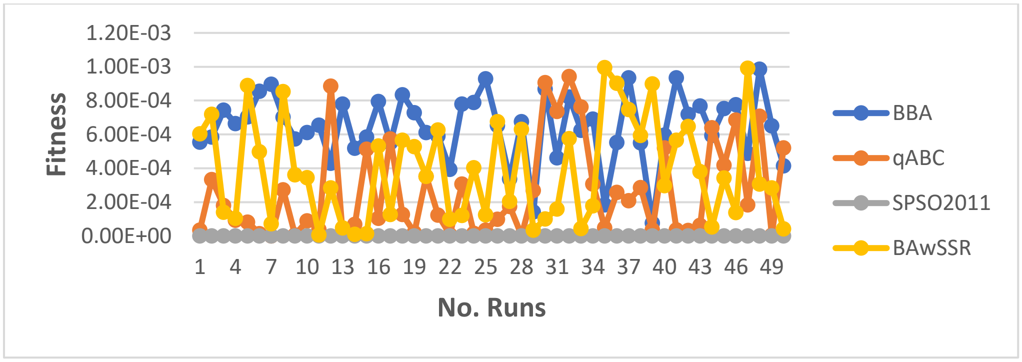
Accuracy figures for 50 runs -Shekel function



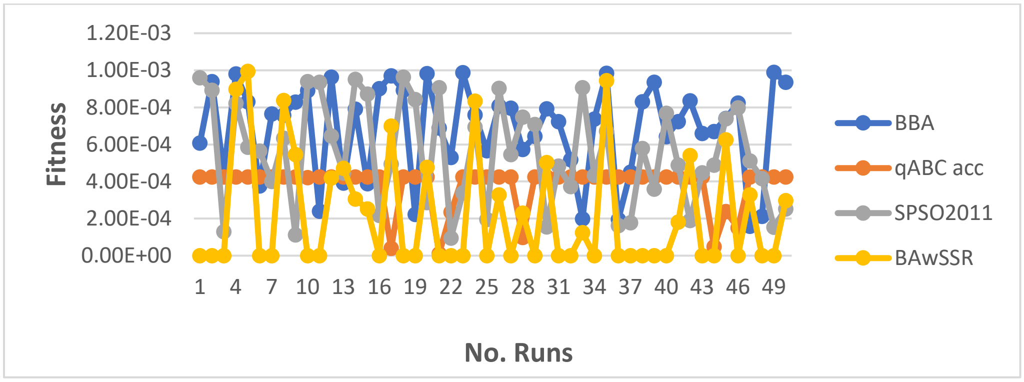
Accuracy figures for 50 runs-Langerman function



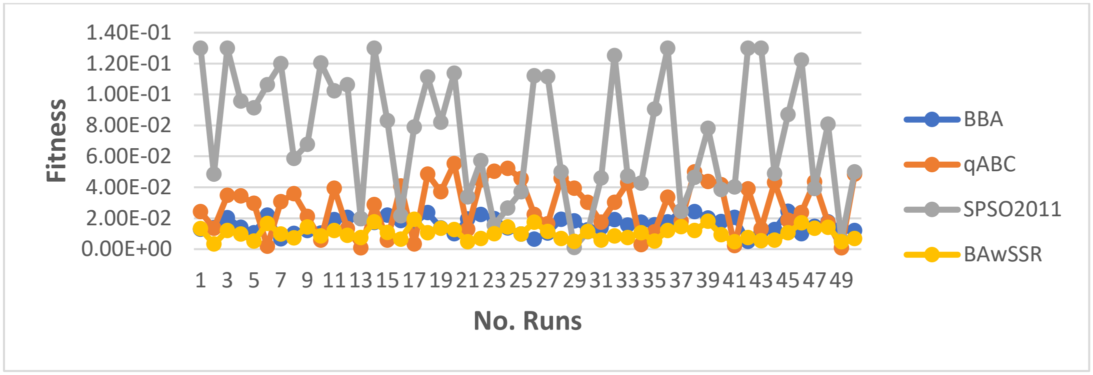
Accuracy figures for 50 runs- Griewank function



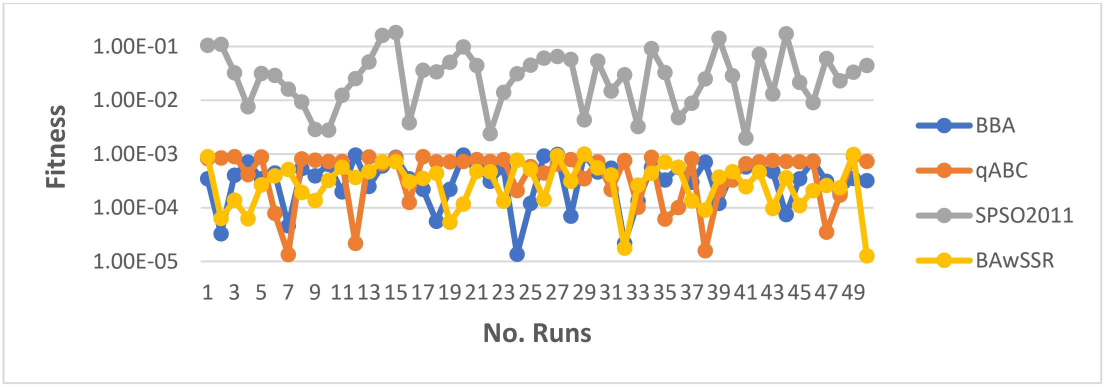
Accuracy figures for 50 runs- Branin function



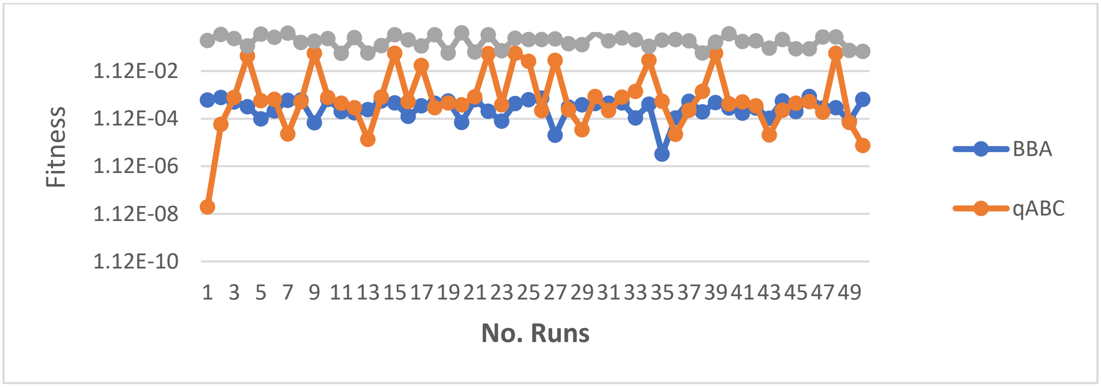
Accuracy figures for 50 runs- Sum of powers function



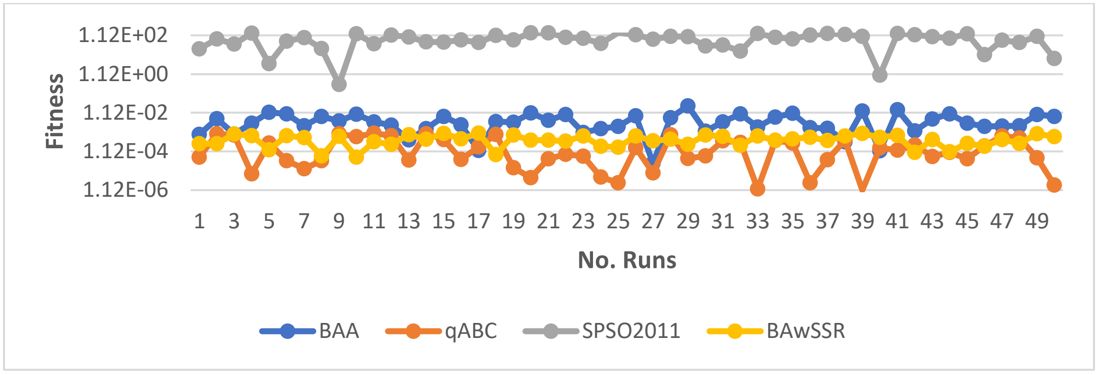
Accuracy figures for 50 runs-Bukin6 function



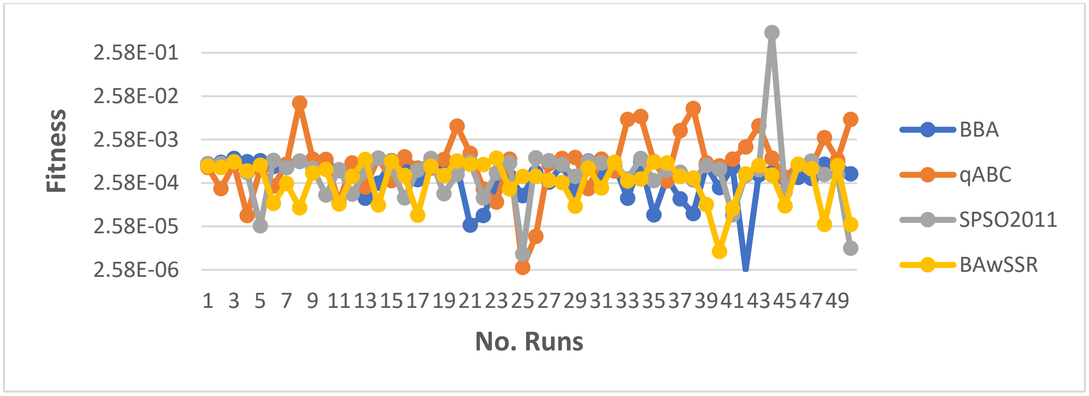
Accuracy figures for 50 runs-Crossit function



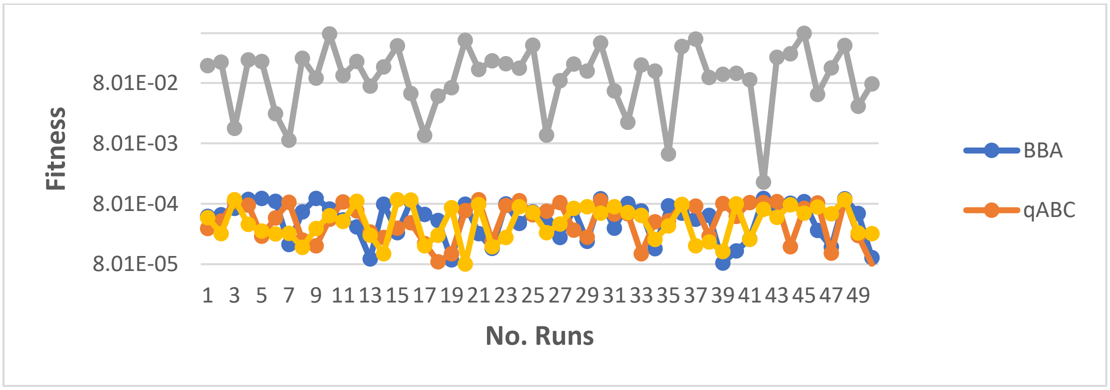
Accuracy figures for 50 runs-Drop function



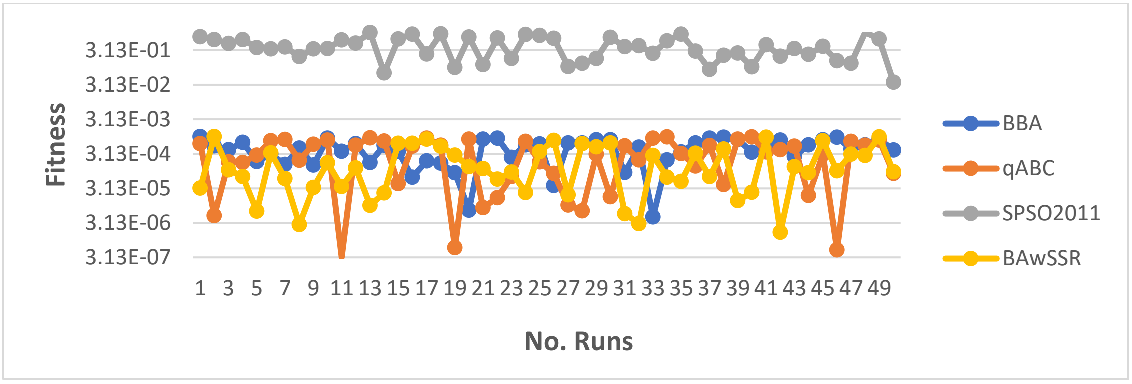
Accuracy figures for 50 runs-Shubert function



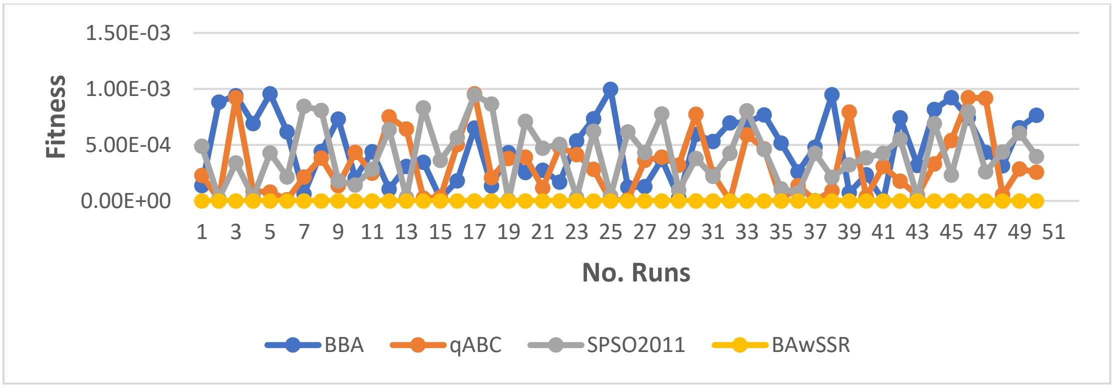
Accuracy figures for 50 runs-Beale



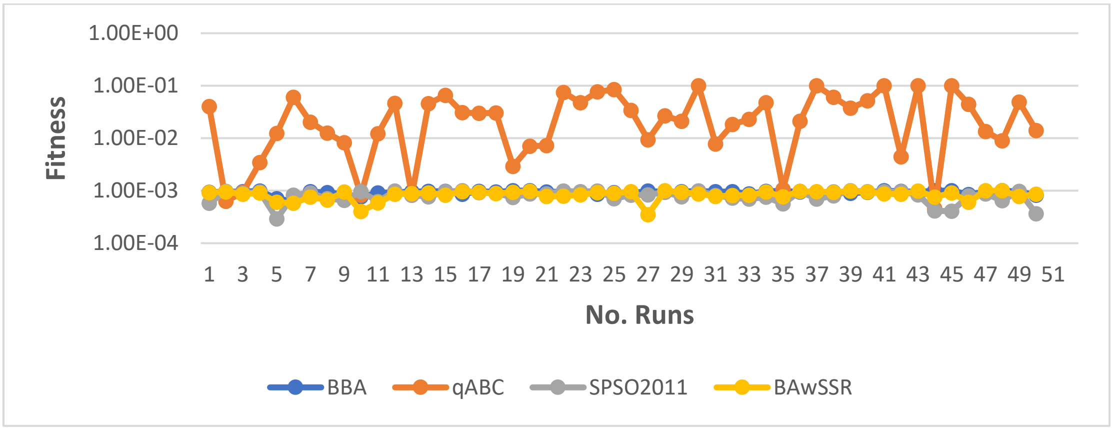
Accuracy figures for 50 runs-McCorm function



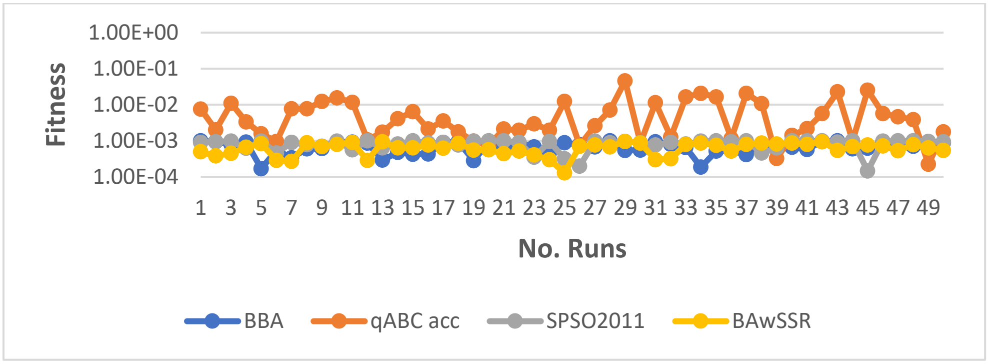
Accuracy figures for 50 runs-Camel6 function



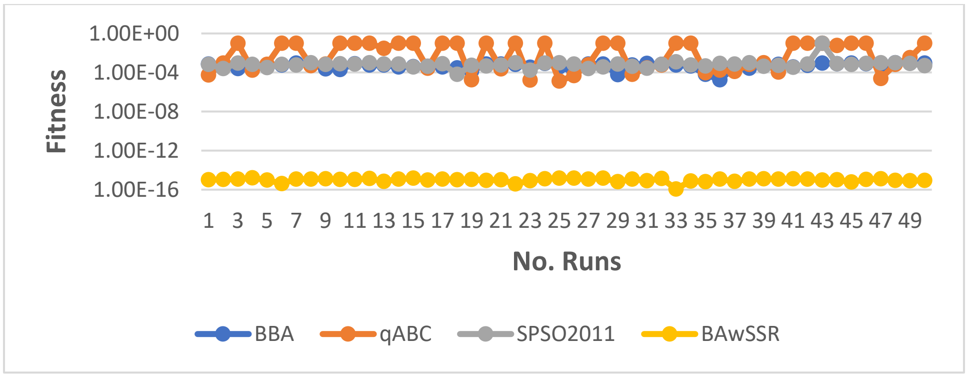
Accuracy figures for 50 runs- Bohachevsk function



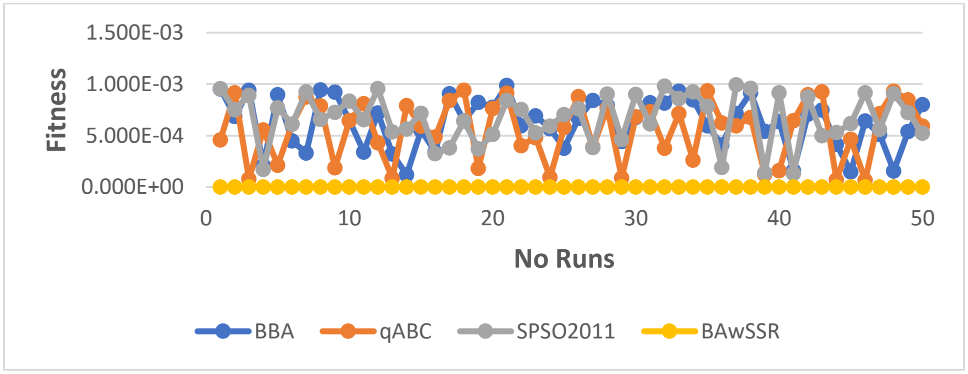
Accuracy figures for 50 runs-Colville function



Accuracy figures for 50 runs- Powersum

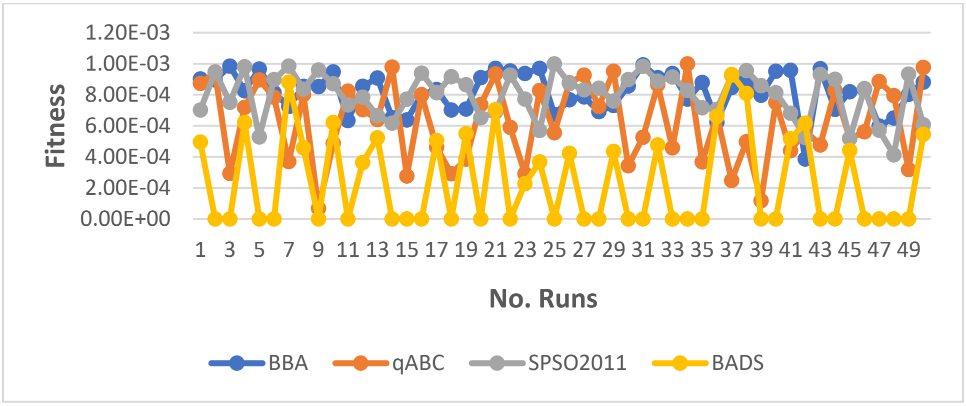


Accuracy figures for 50 runs-Salomon function

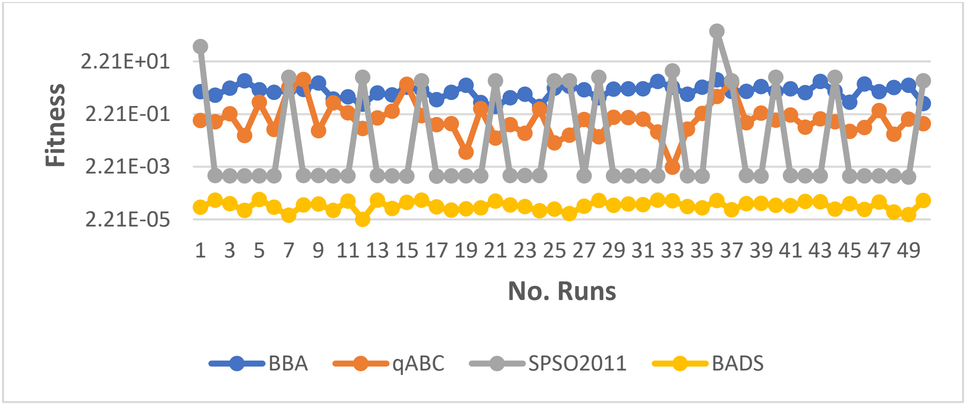


Accuracy figures for 50 runs-Alpine function

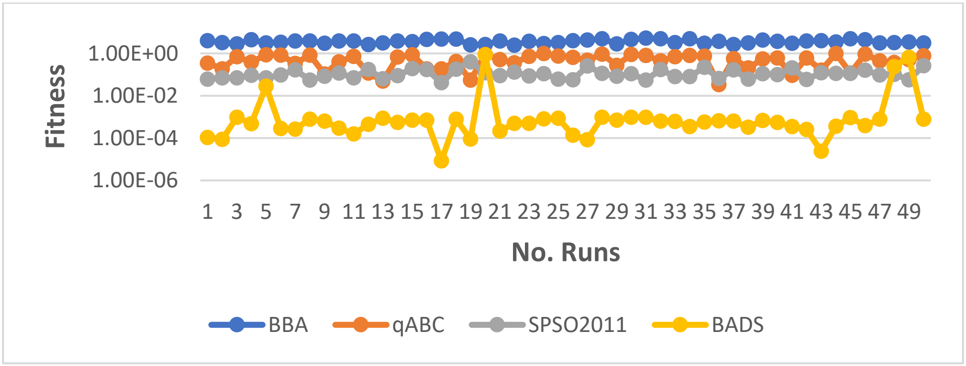
Appendix D List of Charts for BADS Algorithm



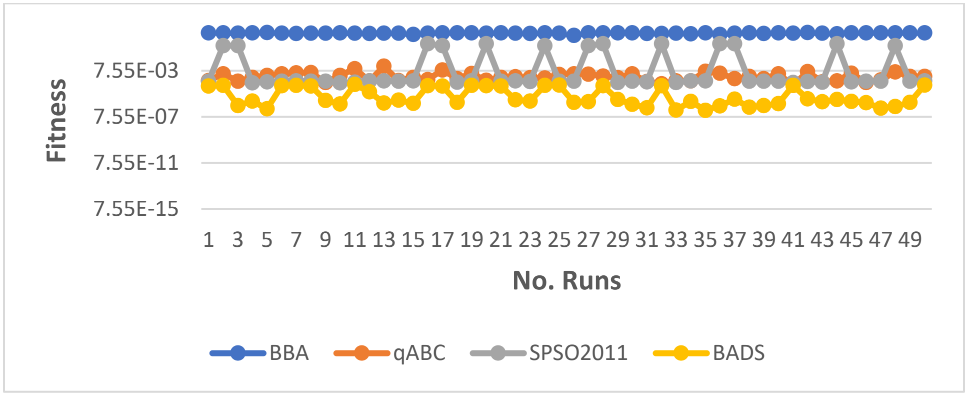
Accuracy figures of 50 runs- Sphere function



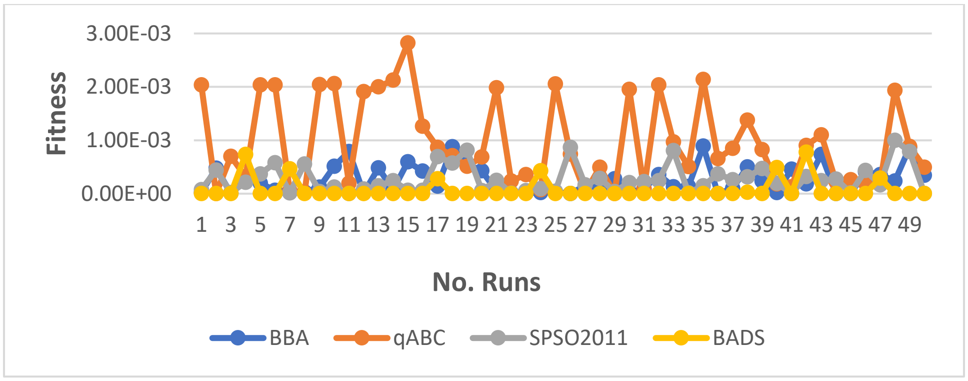
Accuracy figures for 50 runs-Rosenbrock function



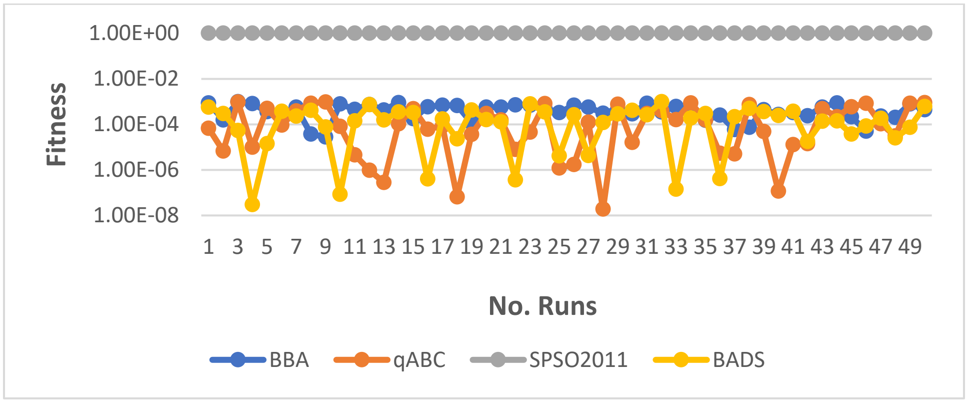
Accuracy figures for 50 runs- Quartic function



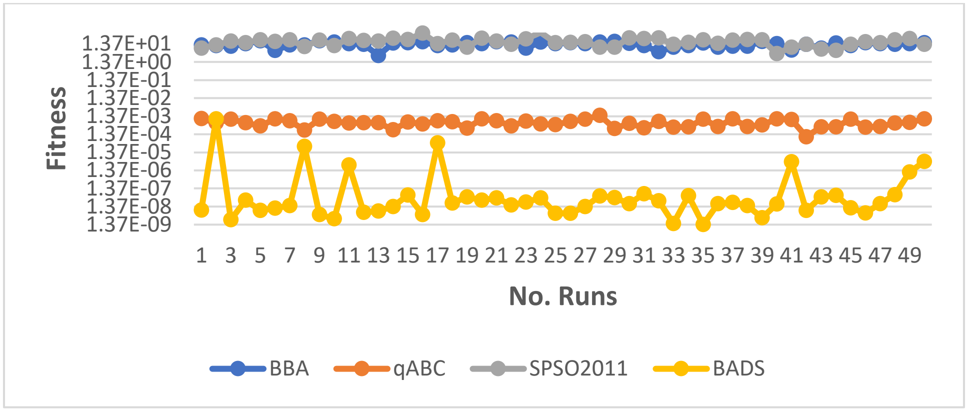
Accuracy figures for 50 runs- Ackley function



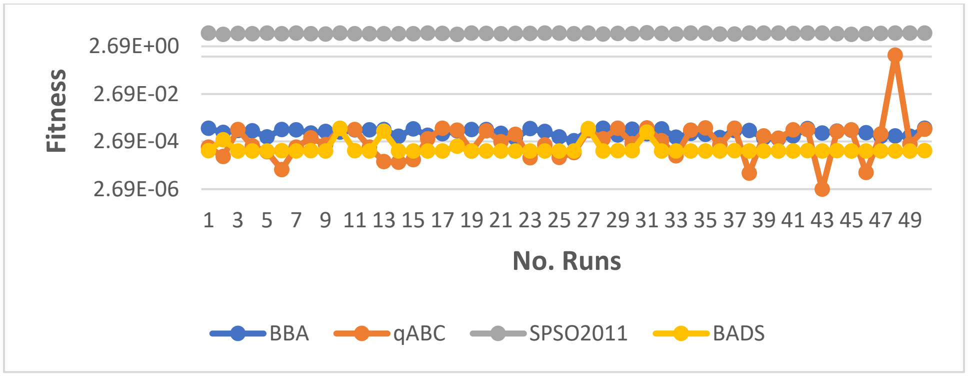
Accuracy figures for 50 runs- Schaffer function



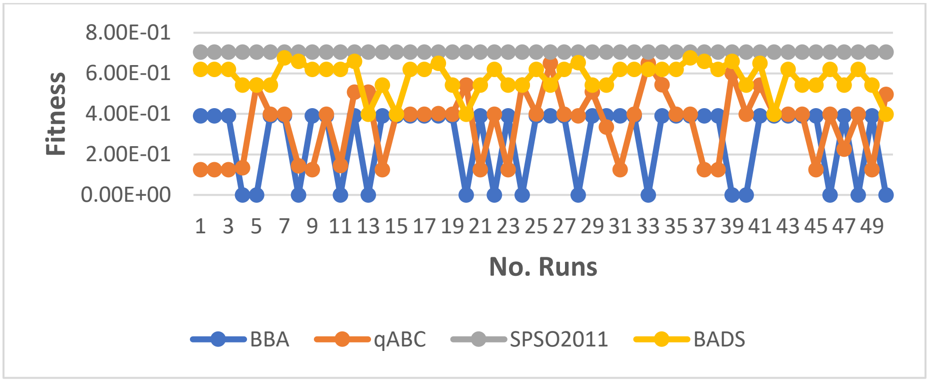
Accuracy figures for 50 runs- Easom function



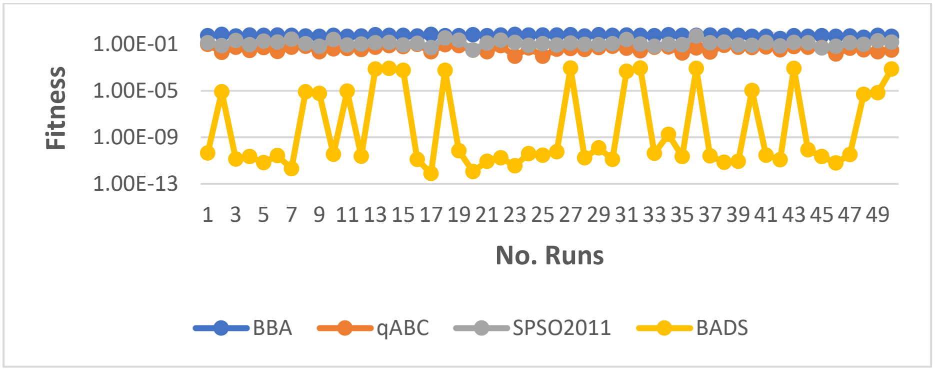
Accuracy figures for 50 runs-Rastrigin function



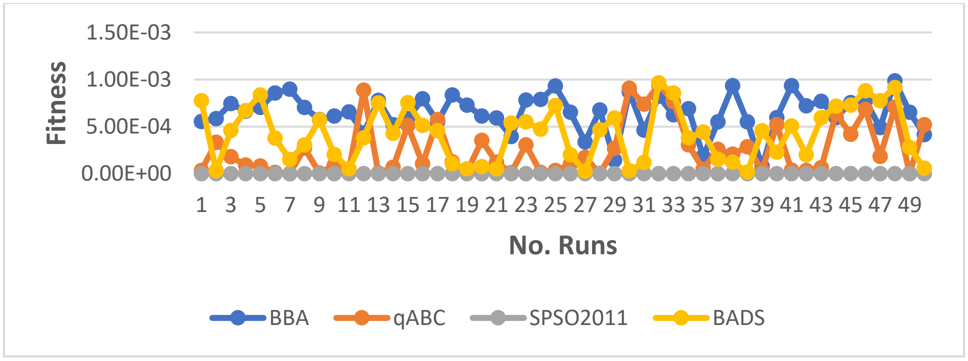
Accuracy figures for 50 runs -Shekel function



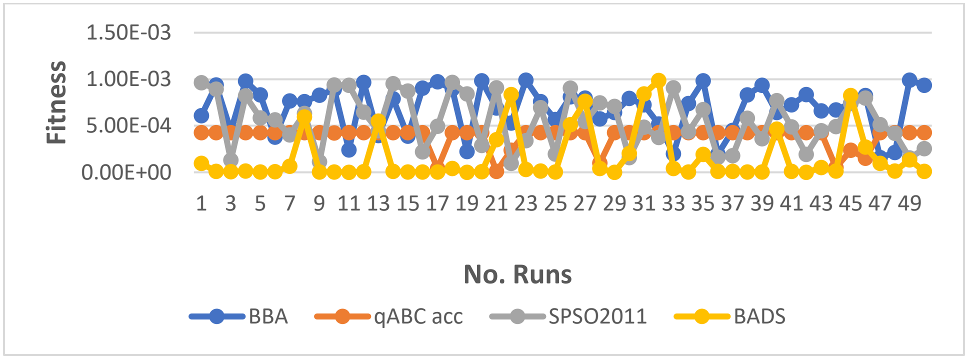
Accuracy figures for 50 runs-Langerman function



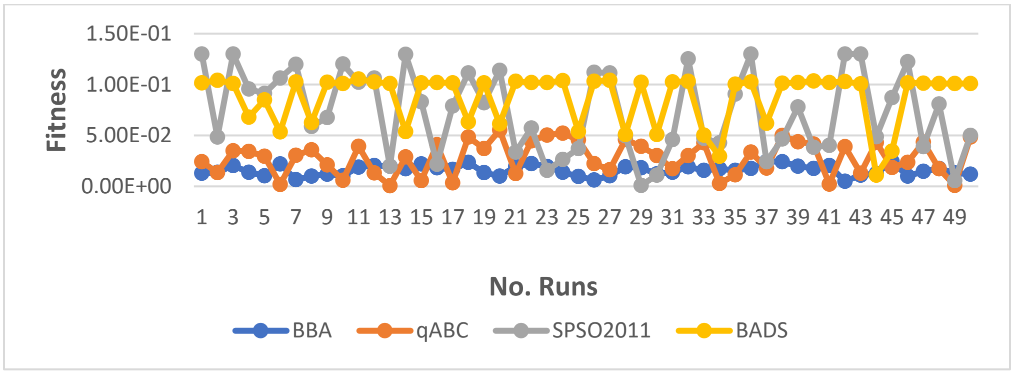
Accuracy figures for 50 runs- Griewank function



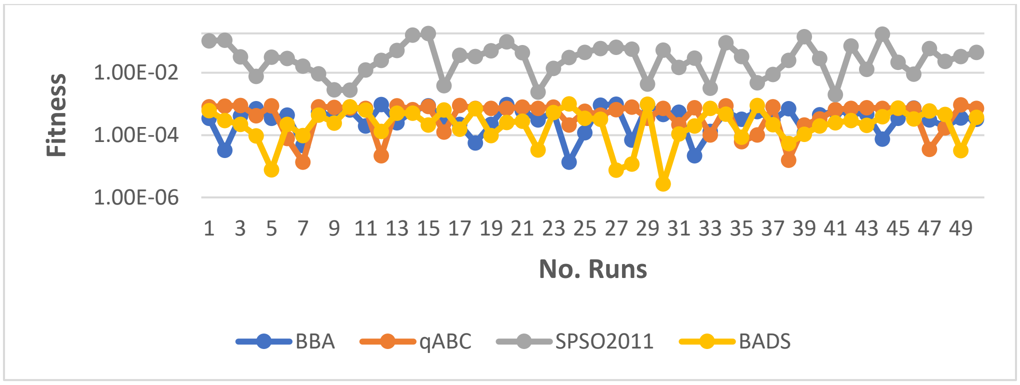
Accuracy figures for 50 runs- Branin function



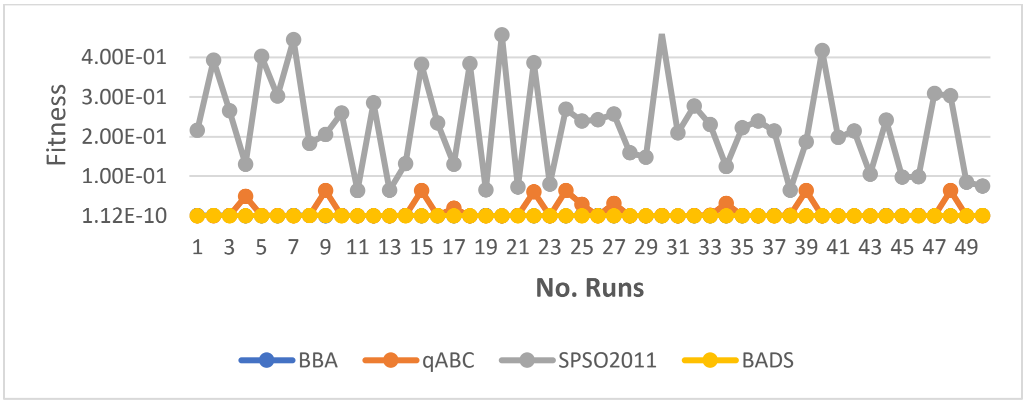
Accuracy figures for 50 runs- Sum of powers function



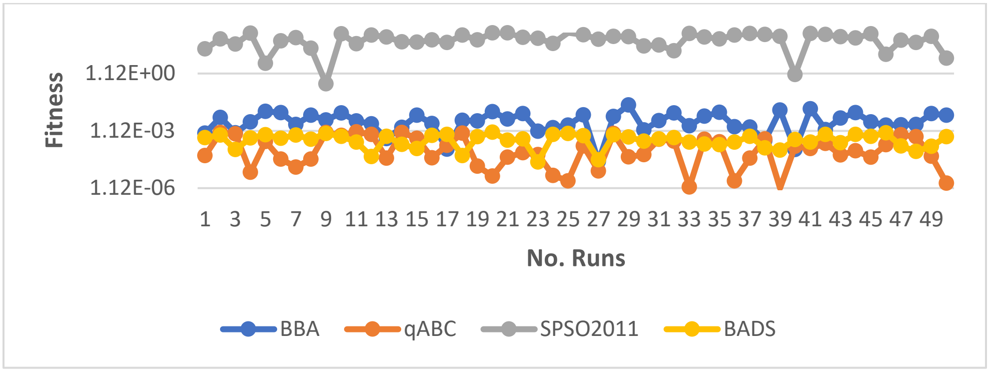
Accuracy figures for 50 runs-Bukin6 function



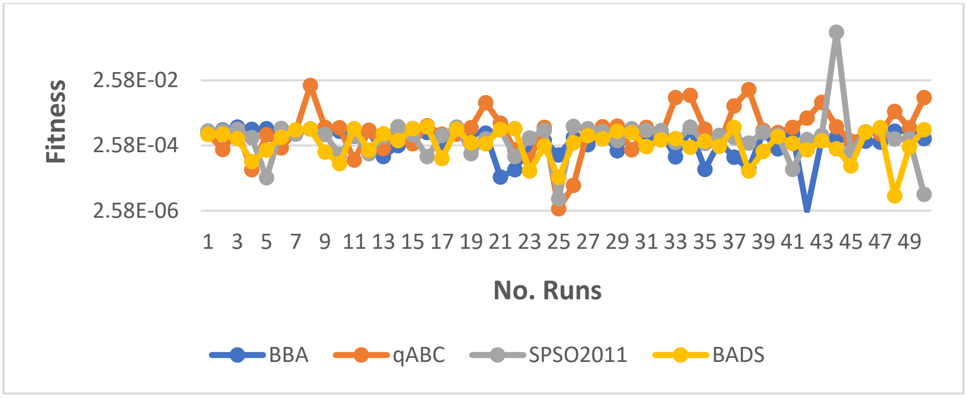
Accuracy figures for 50 runs-Crossit function



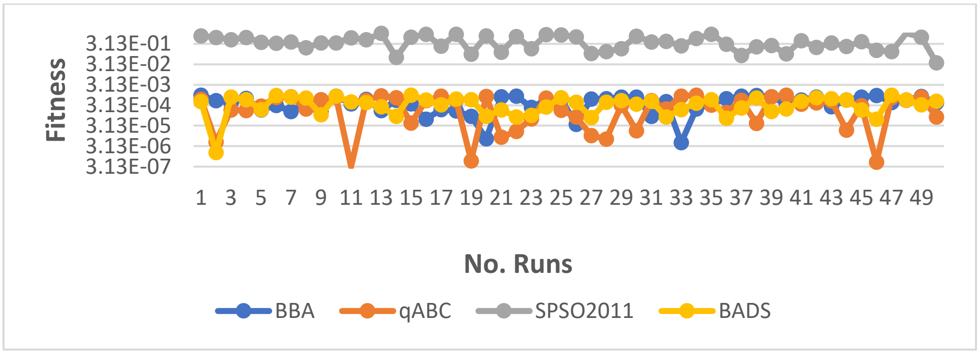
Accuracy figures for 50 runs-Drop function



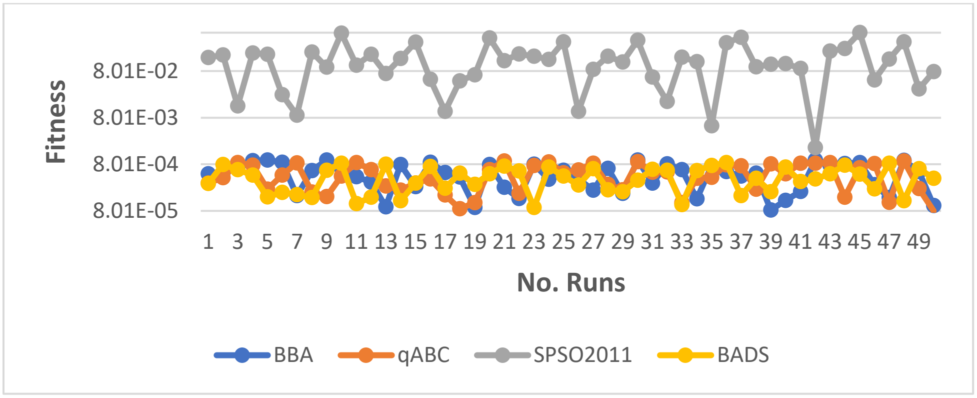
Accuracy figures for 50 runs-Shubert function



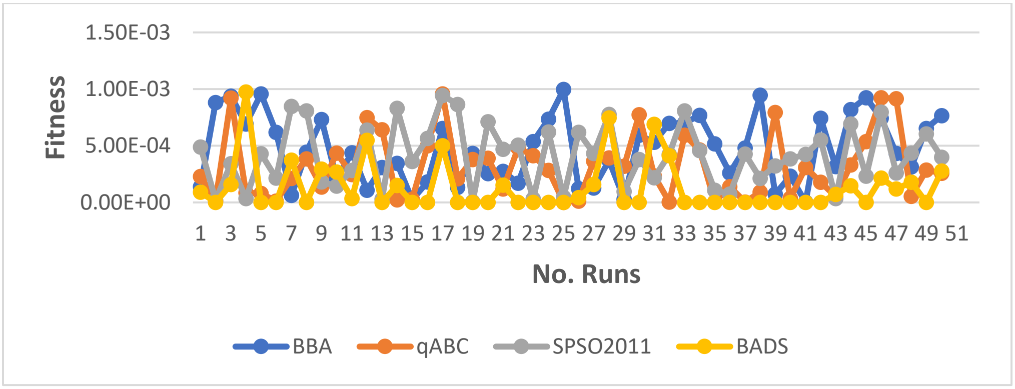
Accuracy figures for 50 runs-Beale function



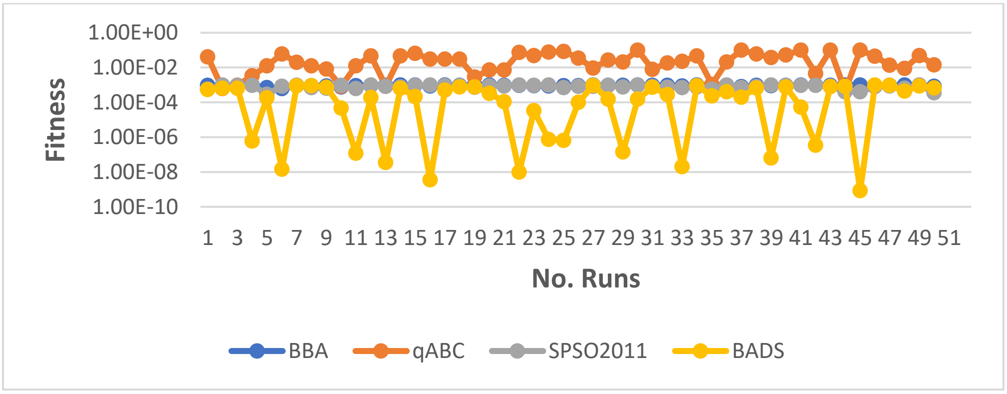
Accuracy figures for 50 runs-Camel6 function



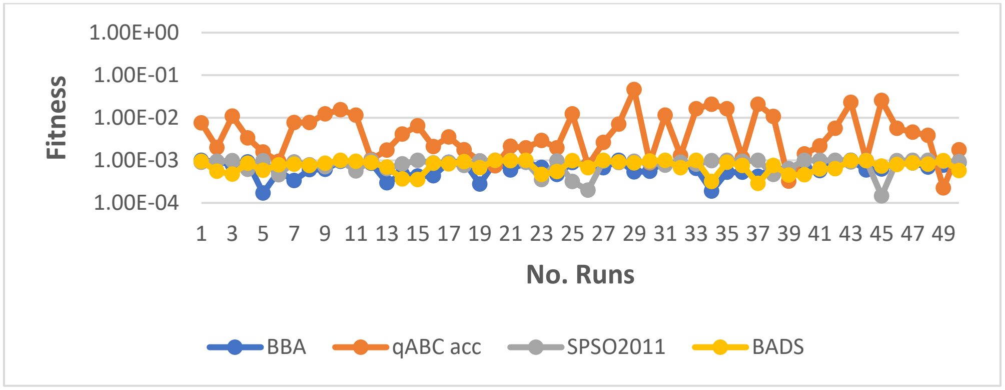
Accuracy figures for 50 runs-McCorm function



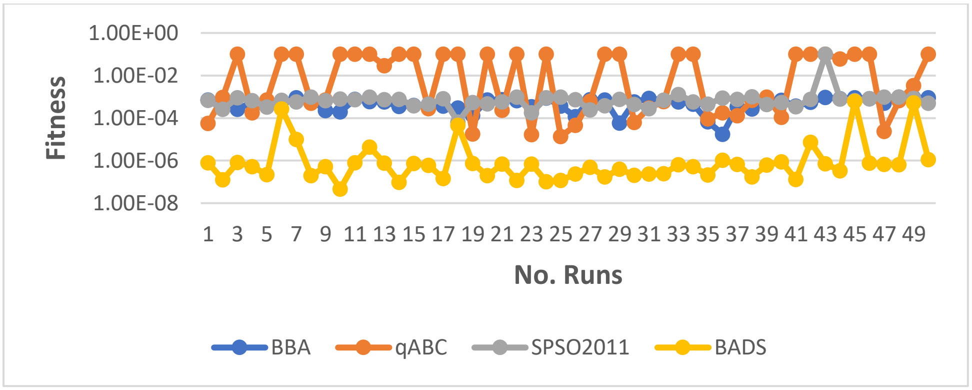
Accuracy figures for 50 runs-Bohachevsky function



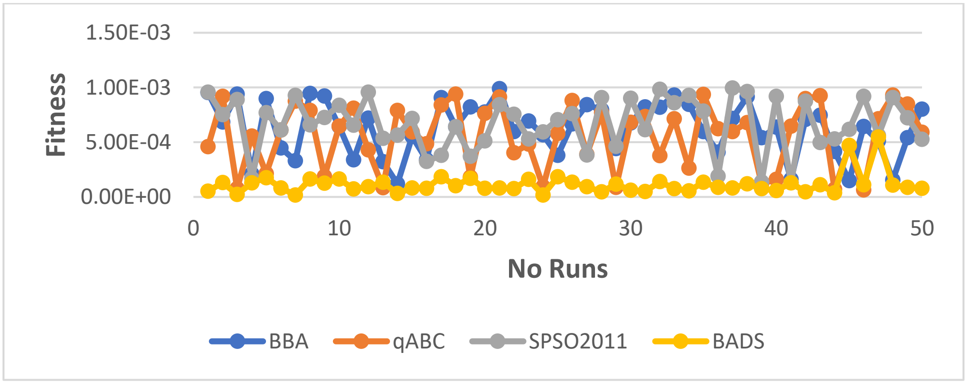
Accuracy figures for 50 runs-Colville function



Accuracy figures for 50 runs- Powersum function

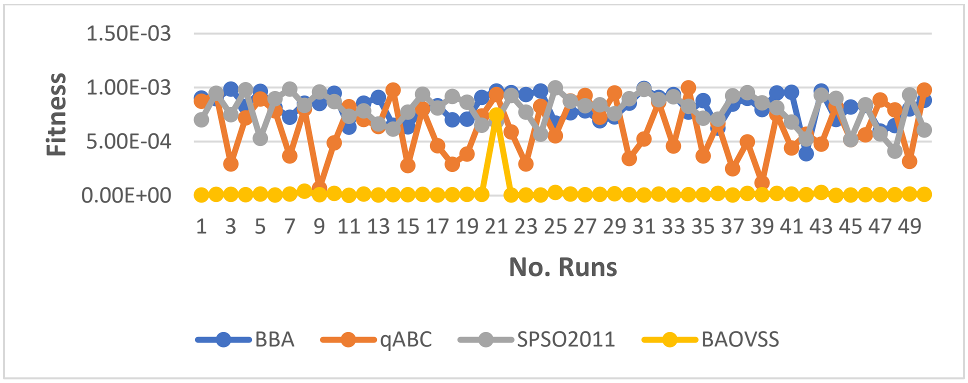


Accuracy figures for 50 runs-Salomon function

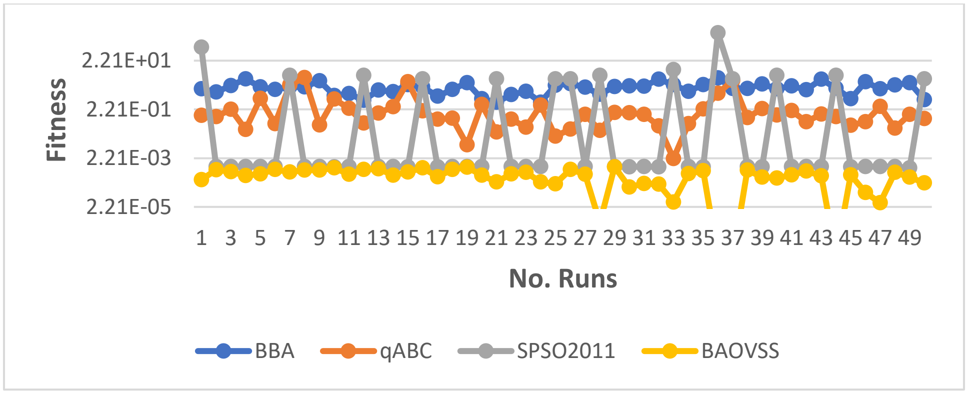


Accuracy figures for 50 runs-Alpine function

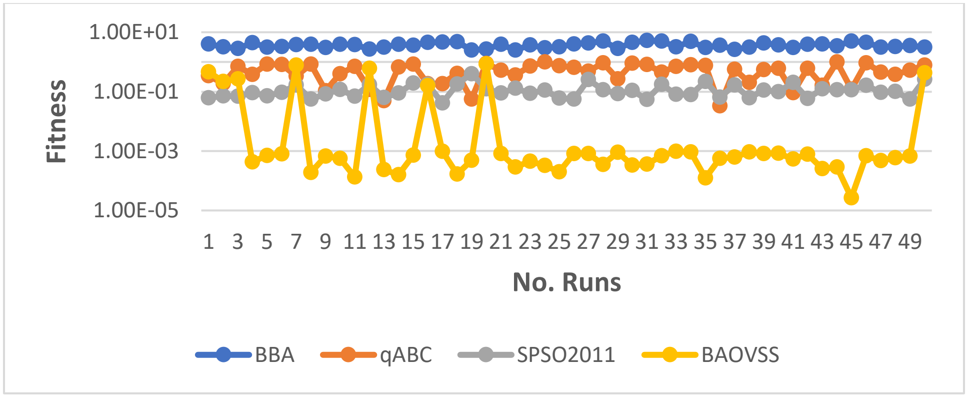
Appendix E LIST OF CHARTS FOR BAOSS ALGORITHM



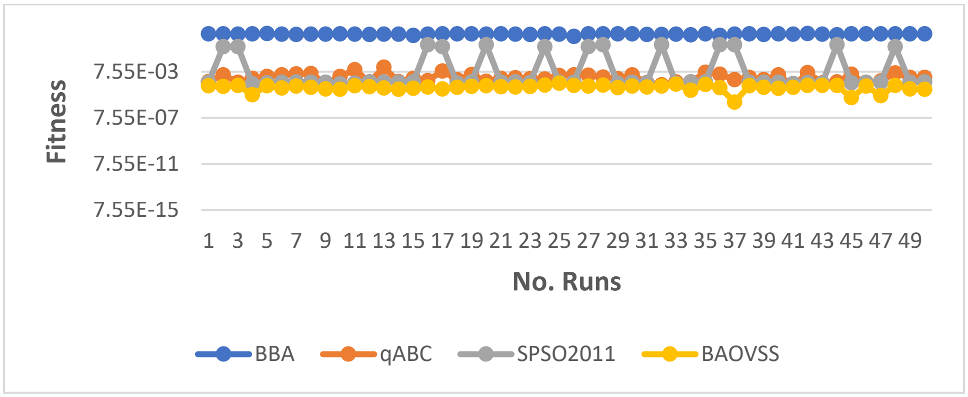
Accuracy figures of 50 runs- Sphere function



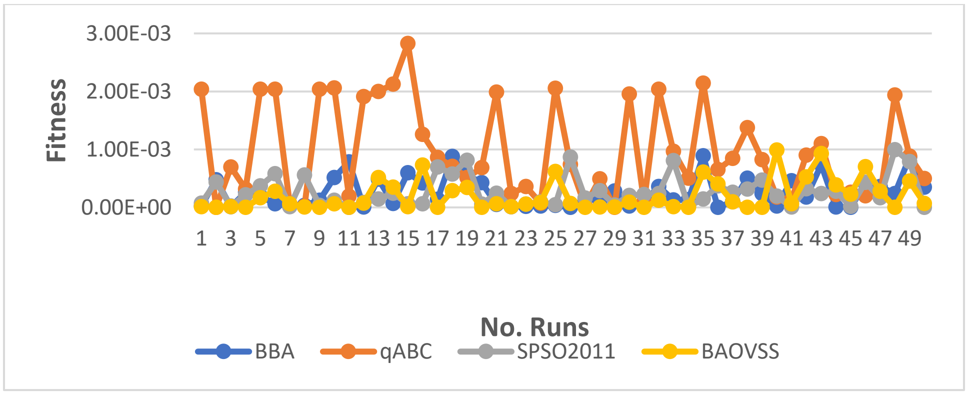
Accuracy figures for 50 runs-Rosenbrock function



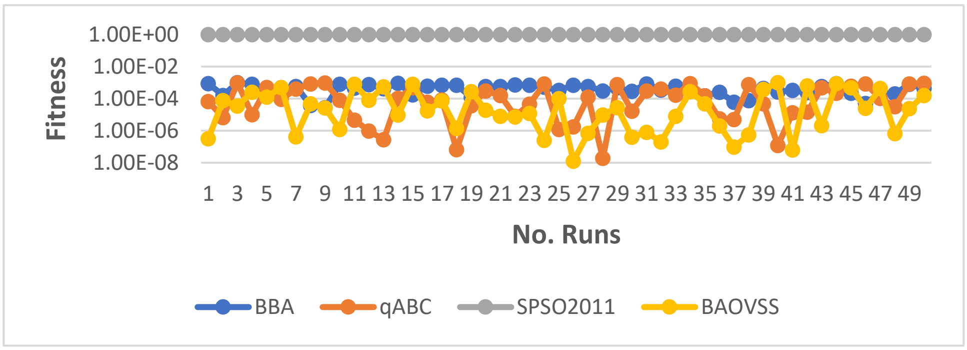
Accuracy figures for 50 runs- Quartic function



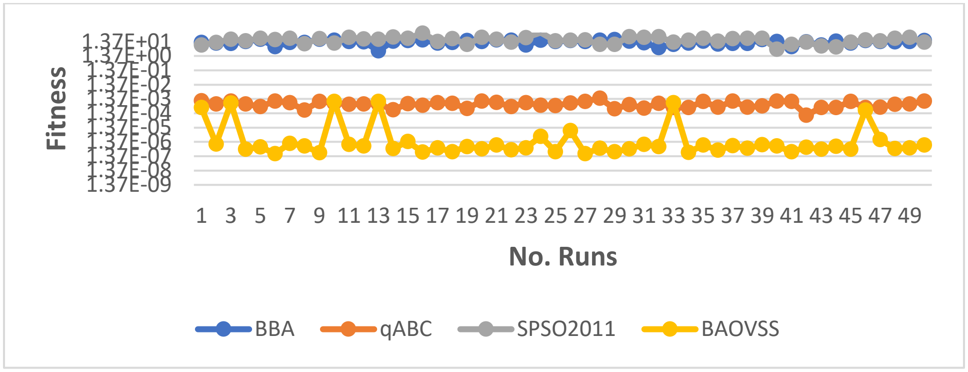
Accuracy figures for 50 runs- Ackley function



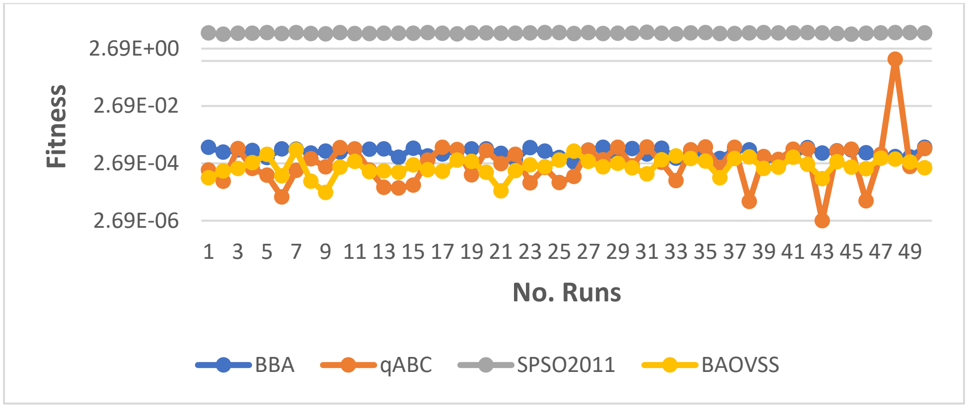
Accuracy figures for 50 runs- Schaffer function



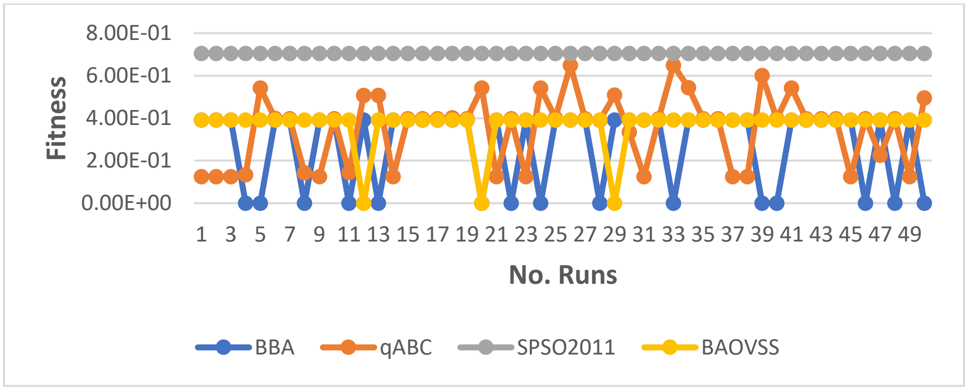
Accuracy figures for 50 runs- Easom function



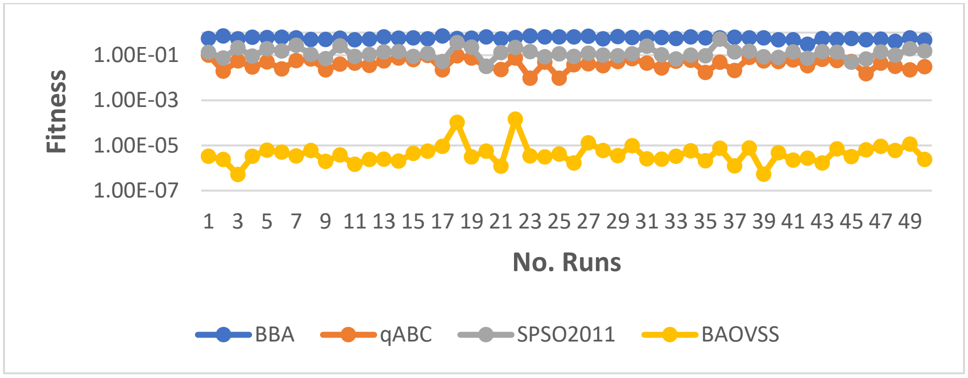
Accuracy figures for 50 runs-Rastrigin function



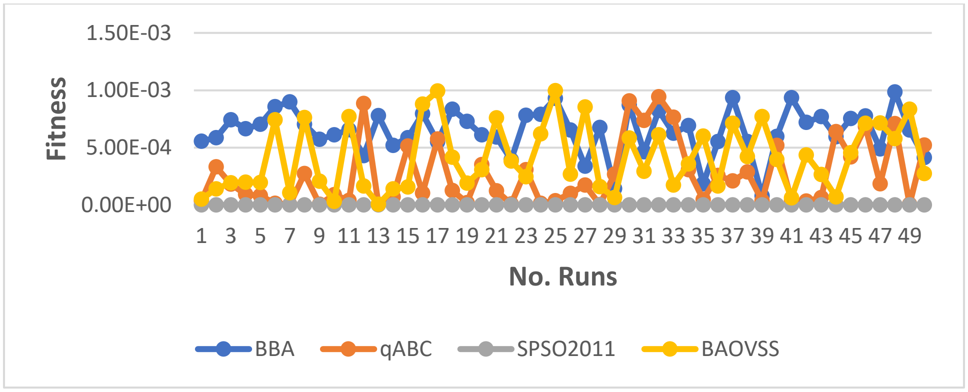
Accuracy figures for 50 runs -Shekel function



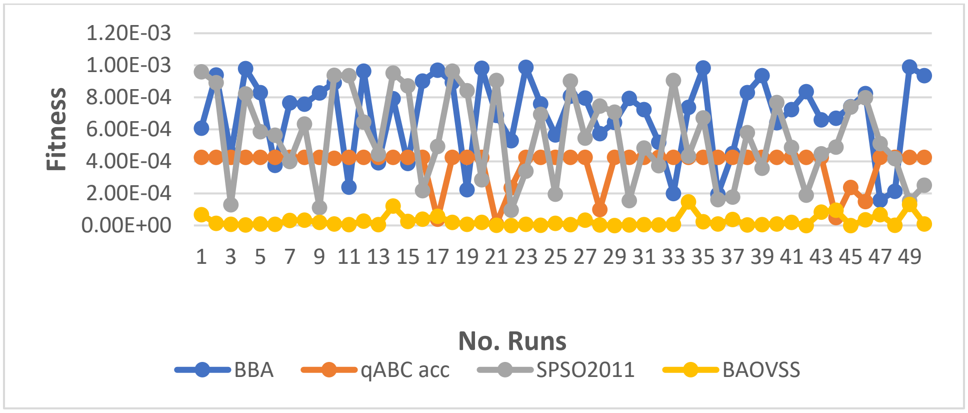
Accuracy figures for 50 runs-Langerman function



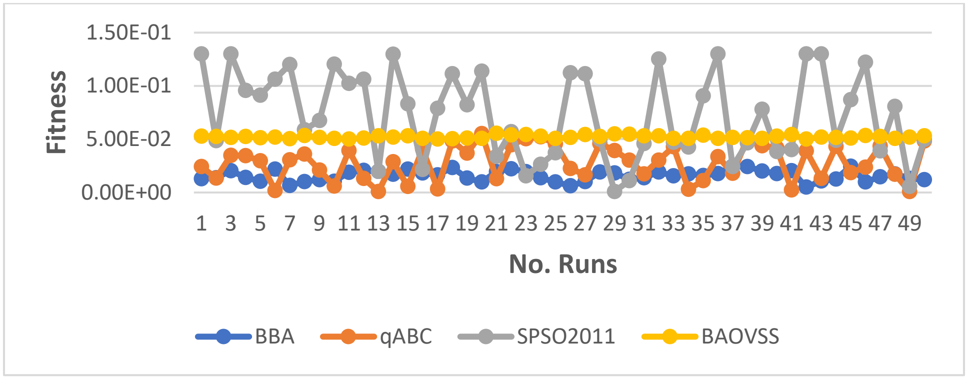
Accuracy figures for 50 runs- Griewank function



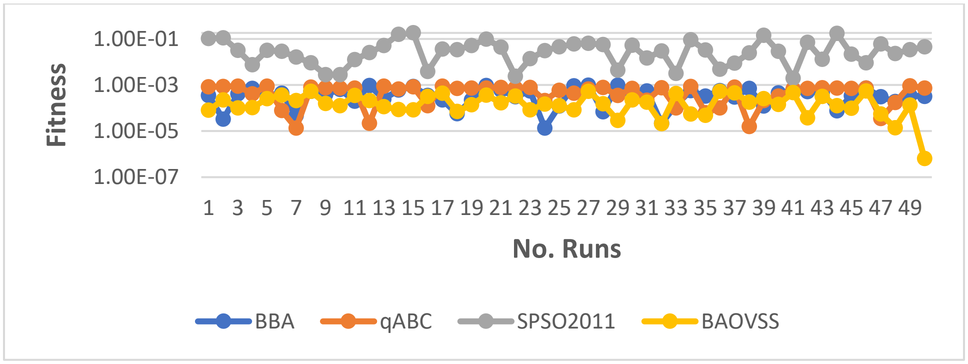
Accuracy figures for 50 runs- Branin function



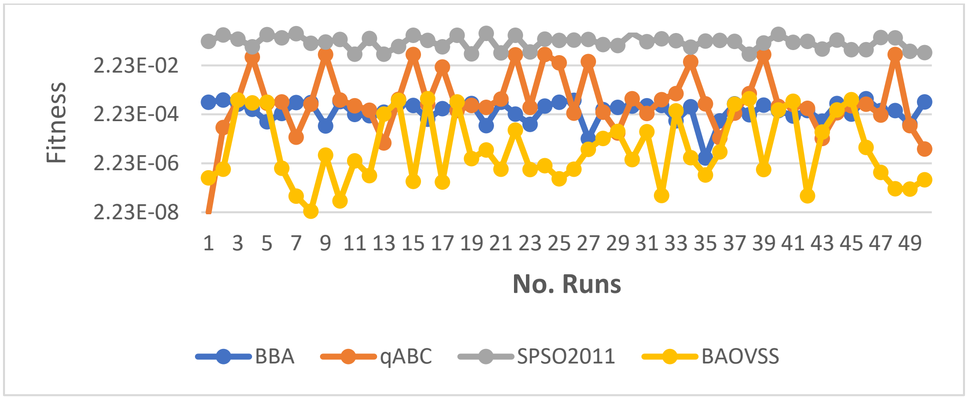
Accuracy figures for 50 runs- Sum of powers function



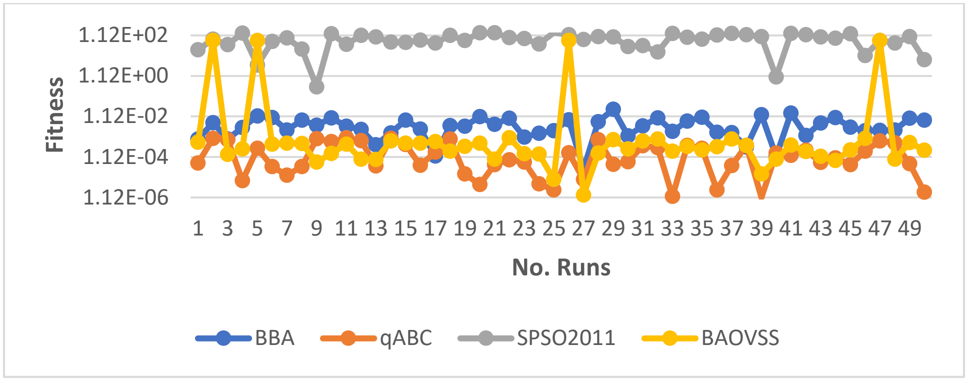
Accuracy figures for 50 runs-Bukin6 function



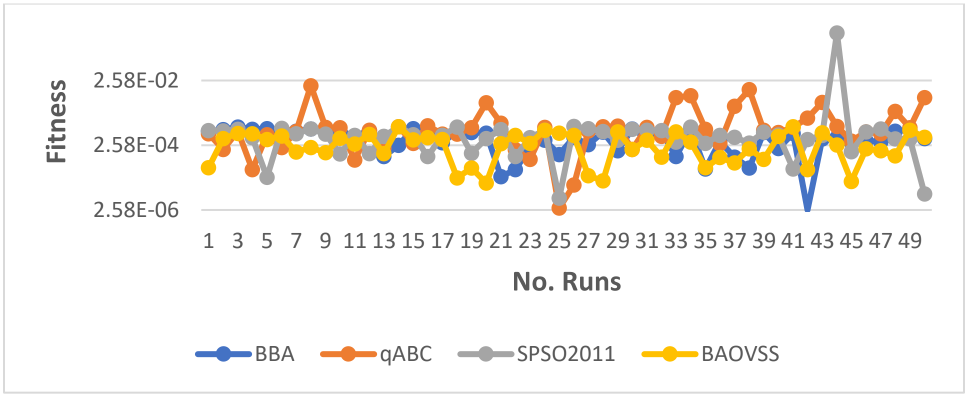
Accuracy figures for 50 runs-Crossit function



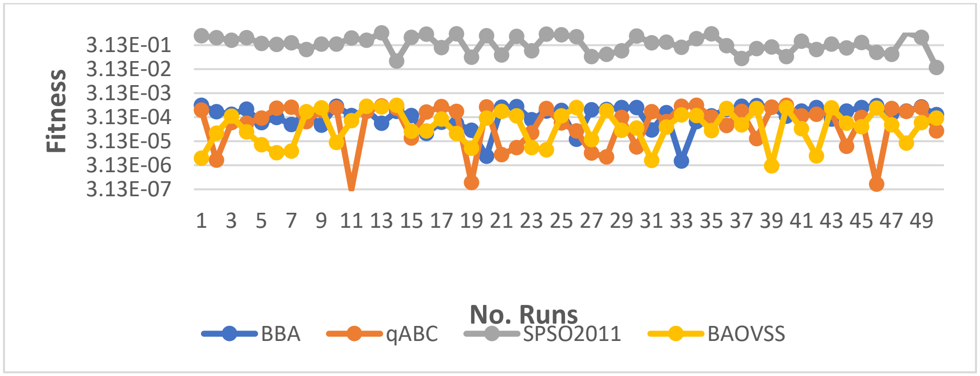
Accuracy figures for 50 runs-Drop function



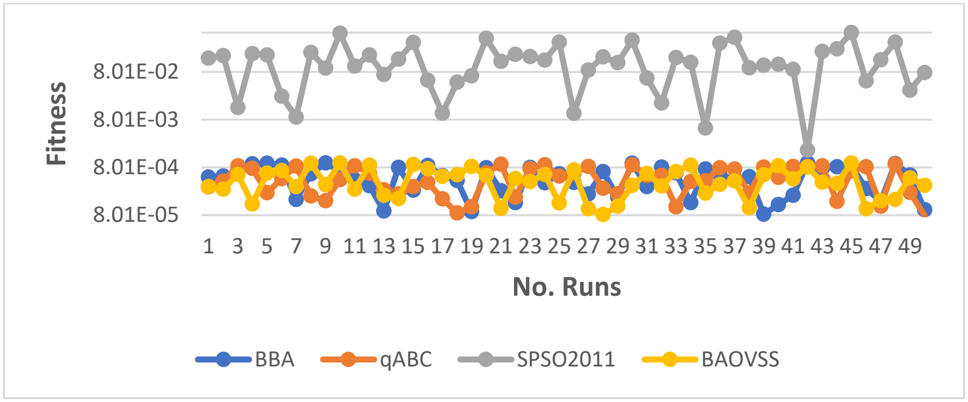
Accuracy figures for 50 runs-Shubert function



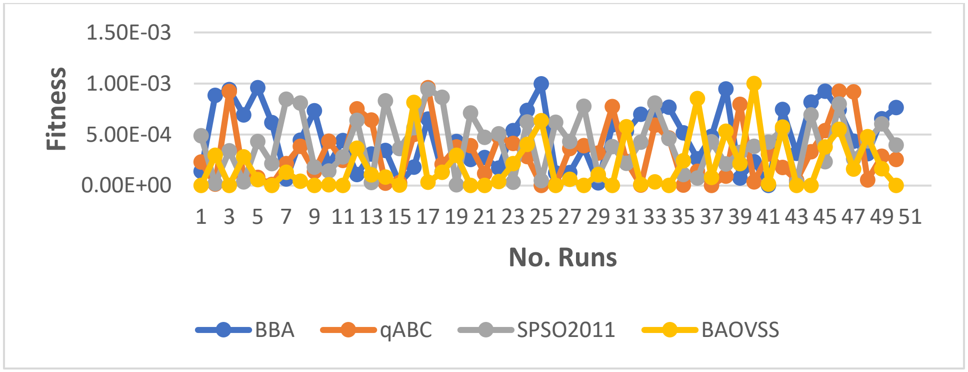
Accuracy figures for 50 runs-Beale



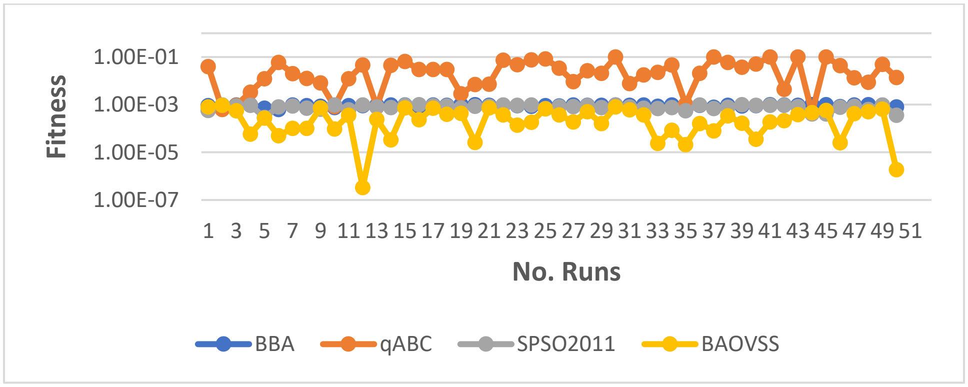
Accuracy figures for 50 runs-Camel6 function



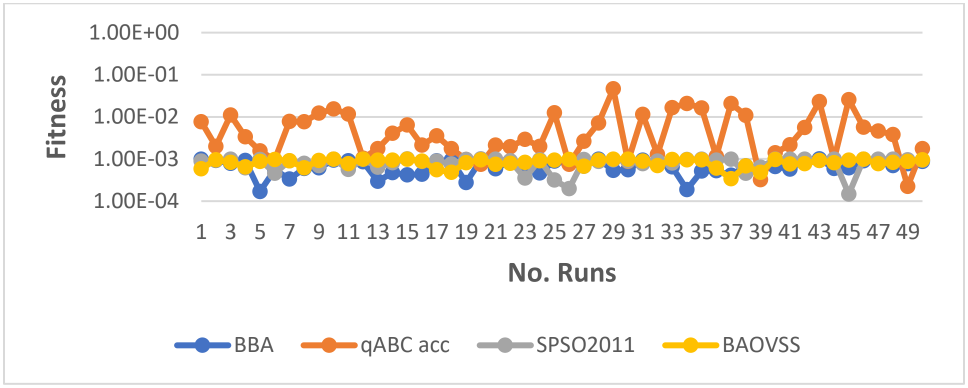
Accuracy figures for 50 runs-Mcorm function



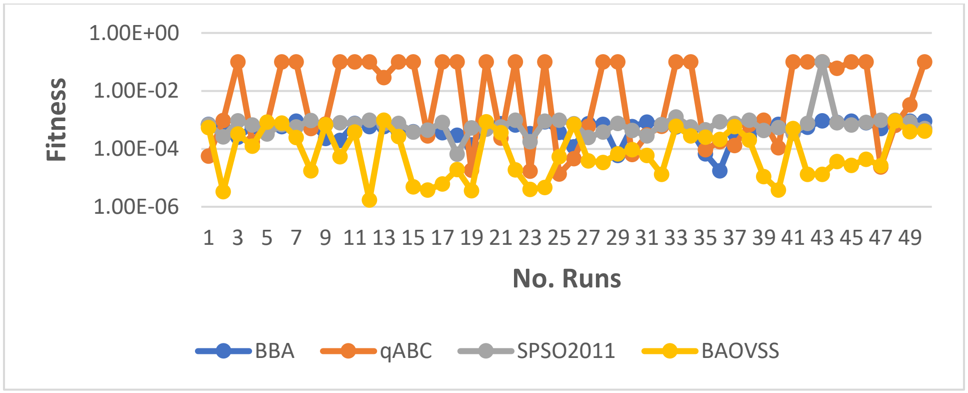
Accuracy figures for 50 runs-Bohachevsky function



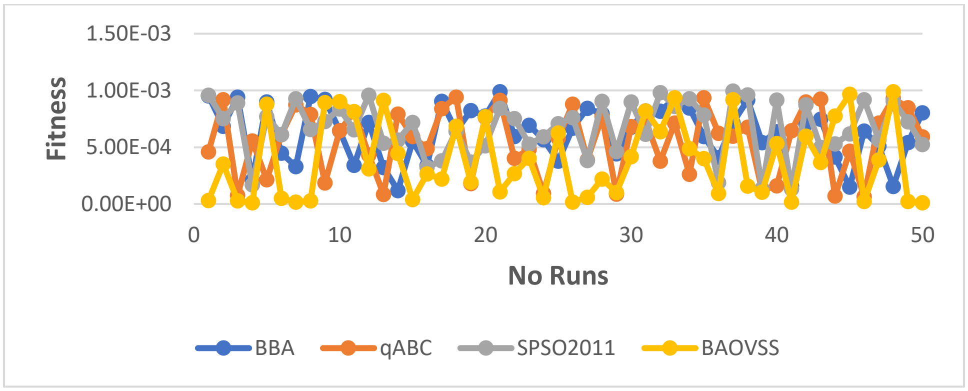
Accuracy figures for 50 runs-Colville function



Accuracy figures for 50 runs- Powersum



Accuracy figures for 50 runs-Salomon function



Accuracy figures for 50 runs-Alpine function

Appendix F LIST OF BENCHMARK FUNCTIONS

Function	Equation	Search range & Minimum
Sphere (10D) $f(x) = \sum_{i=1}^{d} x_i^2$		xᵢ ∈ [-5.12, 5.12] $f(x) = 0$, at x= (0…0)
Rosenbrock (10D)	$\sum_{i=1}^{d}[100(x_{i+1} - x_i^2)^2 + (x_i - 1)^2]$	xᵢ ∈ [-2.048, 2.048] $f(x) = 0$, at x= (1…1)
Quartic(30D)	$f(x) = \sum_{i=1}^{d} i x_i^4 + \text{random}[0,1)$	xi ∈ [−1.28,1.28] $f(x) = 0$ +random noise at x= (0…0)
Ackley (10D)	$f(x) = -a exp\left(-b\sqrt{\frac{1}{d}\sum_{i=1}^{d} x_i^2}\right) - exp\left(-b\sqrt{\frac{1}{d}\sum_{i=1}^{d}\cos(cx_i)}\right) + a + \exp(1)$	a = 20, b = 0.2 and c = 2π xᵢ ∈ [-32.768, 32.768] $f(x) = 0$ at x= (0…0)
Shaffer (2D)	$f(x)=0.5+\frac{\sin^2(x_1^2 - x_2^2) - 0.5}{[1+0.001\,(x_1^2+x_2^2)]^2}$	xᵢ ∈ [-100, 100] $f(x) = 0, at\ x = (0, 0)$
Easom (2D)	$f(x) = -\cos(x_1)\backslash cos\,(x_2) \backslash exp\left(-(x_1 - \pi)^{\{2\}} - (x_2 - \pi)^2\right)$	xᵢ ∈ [-100, 100] $f(x) = -1, at\ x = (\pi, \pi)$
Rastrigin (10D)	$f(x) = 10d + \sum_{i=1}^{d}[x_i^2 - 10\,cos\,(2\pi x_i)]$	xᵢ ∈ [-5.12, 5.12] $f(x) = 0$ at x= (0…0)
Shekel (4D)	$f(x) = -\sum_{i=1}^{m}\left(\sum_{i=j}^{4}(x_j - C_{ji})^2 + \beta_i\right)^{-1}$	xᵢ ∈ [0, 10] $at\ m = 10, f(x) = 10.5364$ at x= (0…0)
Langerman (10D)	$f(x) = \sum_{i=1}^{m} c_i \exp\left(-\frac{1}{\pi}\sum_{j=1}^{d}(x_i - A_{ij})^2\right) \cos\left(\pi\sum_{j=1}^{d}(x_i - A_{ij})^2\right)$	xᵢ ∈ [0, 10] $f(x) = -1.4$
Griewank (10D)	$f(x) = \sum_{i=1}^{d}\frac{x_i{}^2}{4000} - \prod_{i=1}^{d} cos\left(\frac{x_i}{\sqrt{i}}\right) + 1$	xᵢ ∈ [-600, 600] $f(x) = 0$ at x= (0…0)

Branin (2D)	$f(x) = a(x_2 - b{x^2}_1 + cx_1 - r)^2 + s(1-t)cos(x_1) + s$	$x_1 \in [-5, 10]$ $x_2 \in [0, 15]$ $f(x) = 0.37887, at\ x = (-\pi, 12.275), (\pi, 2.275$ $(9.42478, 2.475)$
Sum of Different Powers (Sumpow) (10D)	$f(x) = \sum_{i=1}^{d} \|x_i\|^{i+1}$	$x_i \in [-1, 1]$ $f(x) = 0, at\ xi = (0\ldots0)$
Bukin Function No.6 (Bukin6) (2D)	$f(x) = 100\sqrt{\|x_2 - 0.01{x^2}_1\|} + 0.01\|x_1 + 10\|$	$x_1 \in [-15, -5]$, $x_2 \in [-3, 3]$ $f(x) = 0, at\ x = (-10,1)$
Cross-In-Tray (Crossit) (2D)	$f(x) = -0.0001\left(\left\|sin(x_1)sin(x_2)exp\left(\left\|100 - \frac{\sqrt{{x_1}^2 + {x_2}^2}}{\pi}\right\|\right)\right\| + 1\right)^{0.1}$	$x_i \in [-10, 10]$ $f(x) = -2.06261,$ $at\ x = (1.3491, -1.3491),$ $(1.3491, 1.3491),$ $(-1.3491, 1.3491)$
Drop-Wave function (Drop)2D	$f(x) = -\frac{1 + cos(12\sqrt{{x_1}^2 + {x^2}_2}}{0.5({x_1}^2 + {x^2}_2) + 2}$	$x_i \in [-5.12, 5.12]$ $f(x) = -1, at\ x = (0,0)$
Shubert Function (2D)	$f(x) = \left(\sum_{i=1}^{5} icos((i+1)x_1 + i)\right)\left(\sum_{i=1}^{5} icos((i+1)x_2 + i)\right)$	$x_i \in [-10, 10]$ $f(x) = -186.7309, at = (-7.0835, 4.8580),$ $and\ many\ others$
Beale function (2D)	$f(x) = (1.5 - x_1 + x_1x_2)^2 + (2.25 - x_1 + x_1{x^2}_2)^2 + (2.625 - x_1 + x_1{x^3}_2)^2$	$x_i \in [-4.5, 4.5]$ $f(x) = 0, at\ x = (3, 0.5)$
Six-Hump Camel Function (camel6) (2D)	$f(x) = \left(4 - 2.1{x^2}_1 + \frac{{x^4}_1}{3}\right){x^2}_1 + x_1x_2 + (-4 + 4{x^2}_2){x^2}_2$	$x_1 \in [-3, 3]$, $x_2 \in [-2, 2]$ $f(x) = -1.0316, at$ $x = (0.0898, -0.7126),$ $(-0.0898, 0.7126),$
McCormick function (Mccorm) (2D)	$f(x) = sin(x_1 + x_2) + (x_1 - x_2)^2 - 1.5x_1 + 2.5\,x_2 + 1$	$x_1 \in [-1.5, 4]$, $x_2 \in [-3, 4]$ $f(x) = -1.9133, at$ x $= (-0.54719, -1.54719)$
Bohachevsky function (Boha1) (2D)	$f(x)7 = 7{x^2}_1 + 2x^22 - 0.3cos(3\pi x_1) - 0.4cos(4\pi x_2) + 0.7$	$x_i \in [-100, 100]$ $f(x) = 0, at\ x = (0,0)$
Colville Function (2D)	$f(x) = 100({x^2}_1 - x_2)^2 + (x_1 - 1)^2 + (x_3 - 1)^2$ $+90({x^2}_3 - x_4)^2 + 10.1\,((x_2 - 1)^2 + (x_4 - 1)^2)$ $+19.8(x_2 - 1)(x_4 - 1)$	$x_i \in [-10, 10]$ $f(x) = 0, atx = (1,1,1,1)$
Power Sum Function (Powersum) (4D)	$f(x) = \sum_{i=1}^{d}\left[\left(\sum_{j=1}^{d} {x^i}_j\right) - b_i\right]^2$	$x_i \in [0, 4]$ $f(x) = 0, at = (1,2,2,3)$

Salomon function (2D)	$f(x) = 1 - cos\left(2\pi \sum_{i=1}^{d} {x_i}^2\right) + 0.1 \sum_{i=1}^{d} {x_i}^2$	$x_i \in [-100, 100]$ $f(x) = 0, at = (0,0)$
Alpine N.1 function (Alpine) (2D)	$f(x) = \sum_{i=1}^{d} \lvert x_i \sin(x_i) + 0.1x_i \rvert$	$x_i \in [-10, 10]$ $f(x) = 0, at = (0,0)$