Neural Computing and Applications
https://doi.org/10.1007/s00521-019-04132-w

ORIGINAL ARTICLE

A conceptual comparison of several metaheuristic algorithms on continuous optimisation problems

Absalom E. Ezugwu[1] · Olawale J. Adeleke[2] · Andronicus A. Akinyelu[3] · Serestina Viriri[2]

Received: 18 April 2018 / Accepted: 4 March 2019

Abstract
The field of continuous optimisation has witnessed an explosion of the so-called new or novel metaheuristic algorithms. Though not all of these algorithms are efficient as proclaimed by their inventors, a few of them have proved to be very efficient and thus have become popular tools for solving complex optimisation problems. Therefore, there is a need for a systematic analysis approach to fairly evaluate and compare the results of some of these optimisation algorithms. In this paper, a set of well-known mathematical benchmark functions are compiled to provide an easily accessible collection of standard benchmark test problems for continuous global optimisation. This set of test problems are used to investigate the computational capabilities and the microscopic behaviour of twelve different metaheuristic algorithms. The required number of function evaluations for reaching the best solution and the run-time complexity of the algorithms are compared. Furthermore, statistical tests are conducted to validate the concluding remarks.

Keywords Metaheuristics · Population-based metaheuristics · Swarm intelligence · Continuous domain optimisation

1 Introduction

Many real-world problems may suitably be represented as optimisation problems which are usually very difficult to solve. This difficulty has given rise to the design of several optimisation techniques which are basically developed to find near-optimal solutions. For instance, the meshless computational methods, which solve complex engineering problems without constructing the usually complex mesh points, are gaining ascendancy in literature as optimisation tools. For some recent studies on the applications of meshless methods to optimisation problems, interested readers may look at works in [1–5]. However, even these methods are limited to specific types of engineering optimisation problems. On the other hand, metaheuristic algorithms are general-purpose heuristic methods for solving very complex optimisation problems. In the last decade, several nature-inspired (NI) metaheuristic optimisation algorithms have been proposed and employed to solve complex optimisation and real-world problems that could not be solved using approximate or gradient-based optimisation techniques. Though the majority of these algorithms are not efficient as proclaimed by their pioneers, a few of them have proved to be very efficient and successful and thus have become popular tools for solving complex optimisation and real-world problems. However, the problem with most of the proclamations in regard to the effectiveness of these evolving algorithms is that they are not often tested and compared using sufficient test-suites of problem instances. Moreover, in most studies statistical tests for validating performance superiority claims are usually missing. Therefore, to be fair in drawing valid conclusions, computational and statistical tests of such algorithms are required.

In discussing the performance evaluation of metaheuristic algorithms, twelve metaheuristic algorithms are selected and used as representative algorithms in this paper.

✉ Absalom E. Ezugwu
EzugwuA@ukzn.ac.za

[1] School of Computer Science, University of KwaZulu-Natal, King Edward Road, Pietermaritzburg Campus, Pietermaritzburg, KwaZulu-Natal 3201, South Africa

[2] School of Mathematics, Statistics and Computer Science, University of Kwazulu-Natal, Westville Campus, Private Bag X54001, Durban 4000, South Africa

[3] Department of Computer Science and Informatics, University of the Free State, Bloemfontein 9301, South Africa

Published online: 13 March 2019

These sets of examined algorithms can further be classified into swarm intelligence (SI) and bio-inspired (BI) algorithms. All SI algorithms are inspired by either the collective behaviour of animals or social insects. SI algorithms are among the most popular and widely used metaheuristic methods. One reason for this popularity may be due to the information sharing capability among the multi-agent based structure, which promotes self-organisation, co-evolution and learning during the algorithms' iteration phases. These mechanisms employed by the SI-based algorithms coupled with their parallelisability features inevitably assist in providing the high performance efficiency of this class of algorithms. Examples of SI algorithms are particle swarm optimisation (PSO) algorithms, ant colony optimisation (ACO) algorithms, symbiotic organisms search (SOS) algorithms, cuckoo search (CS) algorithms, firefly algorithms (FA), artificial bee colony (ABC) algorithms and bat algorithms (BA). However, the bio-inspired algorithms, which are closely related to the SI-based algorithms, are nature-inspired algorithms that do not directly use the SI behaviours. Some examples of bio-inspired algorithm include genetic algorithms (GA), differential evolution (DE) algorithms, flower pollination algorithms (FPA), and invasive weed optimisation (IWO) algorithms. A more comprehensive list of the algorithms' classification can be found in [6].

In this paper, a collection of thirty-six different test problems is presented and used as benchmark functions to evaluate each of the representative algorithms. Each benchmark function is identified by its own unique properties that are described in terms of continuity, differentiability, convexity, modality and separability properties. The list of the benchmark function selected for numerical evaluation is shown in Table 1.

The test problems presented in this paper are mainly unconstrained optimisation problems, which can offer significantly large set of benchmark problems for testing, evaluating and comparing the performance of any of the aforementioned metaheuristics and other related global optimisation algorithms in the domain of real-world global optimisation. It is noteworthy to mention here that some of the problems are actually motivated by real-world applications [7]. For example, the Ackley's Function potentially has relevance for real-world applications since the free energy hypersurface of proteins is considered to be of a similar, yet less symmetric, shape to it [8].

Generally, global optimisation problems are vital in many application areas such as industry, engineering and business. The main goal of optimisation is to either minimise or maximise certain objective functions. Mathematically, the continuous nonlinear global optimisation problems described in this paper are of the form:

$$\text{minimise } f(x) \text{ subject to } l \leq x \leq u,\ x \in \Omega$$

where x is a continuous variable vector with D real value decision variables, expressed as $\boldsymbol{x} = (x_1, x_2, \ldots, x_D)$ with domain $\Omega \subset \mathbb{R}^D$, and $f(x)$: $\Omega \mapsto \mathbb{R}$ is a continuous real-valued function. In this case, the decision variable x is defined in the closed interval $[l, u]$. The objective function $f(x)$ may have a nonlinear or non-differentiable form, for which there is no classical gradient-based optimisation techniques for obtaining optimal solutions, in which case the non-gradient-based heuristics are used as an alternative solution approach.

In this study, the performances of twelve global optimisation metaheuristic algorithms are examined. These algorithms include GA [9], PSO [10], ACO [11], SOS [12], CS [13], FA [6], ABC [14], BA [15], DE [16], FPA [17], IWO [18] and BeeA [19]. These algorithms are used extensively in practice. The main goal of this work, however, is targeted at carrying out an in-depth empirical study that would provide deeper insight into the characteristics and behaviours of each representative algorithm. In addition to providing a methodology that can be used to adequately evaluate and compare the performances of the different algorithms detailed in Appendix 1, some measures of merit and evaluation techniques different from other existing techniques [20–23] are likewise presented. The implementation platform for each algorithm is the same and all twelve algorithms are tested on the same set of benchmark problems, which are detailed in Appendix 2. Summarily, the technical contributions of this paper include:

(a) Comprehensive comparisons of twelve metaheuristic algorithms. The algorithms are evaluated on a wide range of 36 standard mathematical benchmark problems.
(b) Statistical test analysis of the performance of the twelve representative metaheuristic algorithms. The analysis forms a basis for a good and unbiased evaluation of the performance and effectiveness of the analysed algorithms.
(c) Mathematical description of a list of thirty-six carefully selected well-known standard mathematical benchmark functions. These functions are often used by researchers in the domain of global optimisation to evaluate the performance of their new algorithms. This list (which cannot be easily collated from a single source) provides an easily accessible collection of standard benchmark test problems for continuous global optimisation. The test functions can be adopted as standard problems for evaluating metaheuristic algorithms in continuous optimisation and other related optimisation application domains.

The rest of the paper is organised as follows: the next section presents a brief discussion of some related work

Table 1 Thirty-six benchmark functions

S/n	Test function	Function name	Search space	*D*	Type	Min
1	F1	Beale	[− 4.5, 4.5]	2	MN	0
2	F2	Easom	[− 100, 100]	2	MN	− 1
3	F3	Matyas	[− 10, 10]	2	UN	0
4	F4	Bohachevsky1	[− 100, 100]	2	US	0
5	F5	Booth	[− 10, 10]	2	US	0
6	F6	Michalewicz2	[0, π]	2	MS	− 1.8013
7	F7	Schaffer	[− 100, 100]	2	UN	0
8	F8	Six Hump Camel Back	[− 5, 5]	2	UN	− 1.03163
9	F9	Boachevsky2	[− 100, 100]	2	UN	0
10	F10	Boachevsky3	[− 100, 100]	2	MN	0
11	F11	Shubert	[− 10, 10]	2	MN	− 186.73
12	F12	Colville	[− 10, 10]	4	MN	0
13	F13	Michalewicz5	[0, π]	5	MS	− 4.6877
14	F14	Zakharov	[− 5, 10]	30	UN	0
15	F15	Michalewicz10	[0, π]	30	MS	− 9.6602
16	F16	Step	[− 5.12, 5.12]	30	US	0
17	F17	Sphere	[− 100, 100]	30	US	0
18	F18	SumSquares	[100, 100]	30	US	0
19	F19	Quartic	[− 1.28, 1.28]	30	US	0
20	F20	Schwefel 2.22	[− 10, 10]	30	UN	0
21	F21	Schwefel 1.2	[− 100, 100]	30	UN	0
22	F22	Rosenbrock	[− 30, 30]	30	UN	0
23	F23	Dixon–Price	[10, 10]	30	UN	0
24	F24	Rastrigin	[− 5.12, 5.12]	30	MS	0
25	F25	Griewank	[− 600, 600]	30	UN	0
26	F26	Ackley	[32, 32]	30	MN	0
27	F27	Drop wave	[− 5.12, 5.12]	2	MN	− 1
28	F28	Hartmann3	[0, 1]	3	MN	− 3.86278
29	F29	Hartmann10	[0, 1]	10	MN	− 3.32237
30	F30	Shekel5	[0, 10]	4	MN	− 10.1532
31	F31	Shekel7	[0, 10]	4	MN	− 10.4029
32	F32	Shekel10	[0, 10]	4	MN	− 10.5364
33	F33	Branin problem	[− 5, 10]	2	MS	0.397887
34	F34	Goldstein–Price	[− 2, 2]	2	MN	3
35	F35	Salomon5	[− 100, 100]	5	MN	0
36	F36	Salomon10	[− 100, 100]	10	MN	0

D dimension, *M* multimodal, *N* non-separable, *U* unimodal, *S* separable

and motivational highlights of the selected candidate algorithms in terms of their algorithmic advantages and disadvantages. Section 3 briefly describes the main methodology procedures used to obtain the performance data for the twelve demonstrative algorithms. Section 4 presents empirical performance comparisons of all the representative algorithms on the thirty-six continuous benchmark functions, which is followed by a comprehensive discussion of the final results and statistical analysis. Section 5 concludes with final remarks. Appendix 1 provides a detailed review of the algorithmic conceptualisation of the twelve global optimisation metaheuristics, whose performances are evaluated and compared. Appendix 2 provides the mathematical description of all the selected thirty-six test problems.

2 Related work and motivation for representative algorithms

This section first examines some of the related work that motivated the current study. A brief description of the main achievements recorded by the existing comparative methods in the literature are highlighted, as well as the major

drives for the current study. The section also presents some of the key advantages, disadvantages, and design inspiration of the representative algorithms based on biological motivations.

2.1 Related work

There are truly several basic and hybrid variants of popular stochastic metaheuristic algorithms that have shown potentials to solve various complex continuous and combinatorial optimisation problems. It is noteworthy to mention here that the term popular metaheuristics is used to denote specifically those sets of stochastic nature-inspired algorithms that have achieved notable track records in terms of their capabilities to solve real-world problems that span across different global optimisation research domains [24–26]. Furthermore, it is equally important to highlight at this point that it is also extremely difficult to exhaustively and systematically analyse the relative merits of these recent proposed global optimisation algorithms. However, there are few records in the literature that have attempted to present some level of relative comparison for some of these algorithms [21–23, 27]. Subsequently, a brief discussion of some related work is presented.

Ali et al. [27] presented a comprehensive numerical evaluation of five stochastic algorithms on selected continuous global optimisation test problems. The five evaluated algorithms include improving hit-and-run, hide-and-seek, controlled random search, real coded genetic algorithms, and differential evolution. In the work of Ali et al. [27], a performance profile plot based on the improvement in objective function values was constructed to investigate the macroscopic behaviour of each of the candidate algorithms. The authors carried out other similar investigations on the microscopic behaviour of the selected algorithms through the construction of quartile sequential plots, and subsequently made contrasts based on the information gained from these plots. Their experimental evaluation steps hinged on using three maximum number of function evaluations ($100n^2$, $10n^2$, and $10n$, respectively, where n denotes problem dimension) to execute each of the algorithms after several experimental trials. The evaluation results, which were based on the exploration of the length of run using the three carefully selected maximum number of function evaluations, revealed that this approach significantly affected the performance of the individual algorithms.

In Ma et al. [21], the study of the characteristics of various popular evolutionary algorithms were conducted. In their study, the authors compared the basic and advanced versions of GA, biogeography-based optimisation (BBO), DE, evolution strategy (ES) and PSO to explore their optimisation ability on a set of real-world continuous optimisation problems. Furthermore, the conceptual discussion on the equivalences of the GA, BBO, DE, ES and PSO was presented. One of the findings of the experimental results revealed that under certain experimental conditions the classical versions of BBO, DE, ES and PSO are equal to the GA with global uniform recombination (GA/GUR). However, their main contribution shows that the conceptual equivalence of the algorithms is supported by the fact that algorithmic modifications result in very different performance levels. Therefore, this major finding clearly suggests the need for further investigation into the critical analysis and evaluation of the performances of the various proposed metaheuristic algorithms as presented in this current study.

Similarly, in Ma et al. [22], a comparative study was presented on the algorithmic equivalence of PSO and various other newer swarm intelligence algorithms, including the shuffled frog leaping algorithm (SFLA), the group search optimiser (GSO), the FA, ABC and gravitational search algorithm (GSA). The author numerically compared the aforementioned algorithms with their basic and advanced versions on some continuous benchmark functions and combinatorial knapsack problems. Again, it was discovered in the study by Ma et al. [22] that the standard versions of SFLA, GSO, FA, ABC, and GSA, are all algorithmically identical to the basic PSO under certain experimental conditions. The empirical results also revealed the performance discrepancies between the basic and advanced version of the compared algorithms on the two choice problems used. More so, the advanced version of ABC performs best on the continuous benchmark functions, and advanced versions of SFLA and GSA perform best on the combinatorial knapsack problems.

In the study presented by Civicioglu and Besdok [23], the results of the conceptual comparison of the CS, PSO, DE and ACO were analysed. The run-time complexity and the required number of function evaluations for acquiring a global minimiser were used as methods of evaluation. Civicioglu and Besdok [23] compared the numerical optimisation problem solving successes of the four mentioned algorithms and backed claims of their computational result with statistical analysis using over 50 different mathematical benchmark functions. The empirical results from their work revealed that the problem solving success of the CS algorithm can be statistically compared to the DE algorithm or in other words, there was no statistically significant difference between the performance of CS and DE algorithms.

This study is motivated by the recent surge in the development and implementation of several new metaheuristic algorithms by different researchers such as Cheng and Prayogo [12], Yang [28], Yang et al. [29] who claimed to base their drive on the common ground that there is no

single metaheuristic optimisation algorithm that is capable of solving all optimisation problems of different types and structures. Thus, the need to continuously seek to improve the already existing algorithms or develop new metaheuristic optimisation techniques that are capable of solving even more complex and large scale global optimisation problems [23]. Therefore, in line with this common belief, the current study seeks to investigate the algorithmic performance superiority of twelve selected popular metaheuristic algorithms mentioned in Sect. 1 and to also ascertain whether this set of algorithms has any similarity in their performance behaviours under the same experimental conditions. It is equally important to mention here that while existing work used fewer algorithms for their comparisons, this study employed twelve well-known algorithms, among which also include some recently implemented algorithms for its evaluation analysis. More so, the interest in the twelve representative algorithms are based on the fact that CS, FPA, BA, ABC, IWO and BeeA are recent nature-inspired algorithms that have attracted the attention of researchers [30, 31]. Also, ACO, PSO, DE, GA are early established and popular nature-inspired algorithms that have produced remarkable results in the literature [30]. In the next section, a brief highlight of merit and demerit of each representative algorithm is presented to support the choices made in selecting the above-mentioned popular algorithms.

2.2 Representative algorithms: advantages, disadvantages, and motivation

The twelve metaheuristic algorithms discussed above generally have their various strengths and weaknesses. Most of these techniques require much iteration and perform poorly if there is no adequate parameter tuning. Moreover, metaheuristics are designed to search for near-optimal solutions as these do not have the capacity to provide an optimal solution. Nevertheless, metaheuristics have some advantages that can be applied to a wide range of problems, either independently or in combination with other traditional techniques. Moreover, most metaheuristics have mechanisms for information sharing, which are responsible for enhancing quick convergence [32]. Further, NI inspired optimisation techniques provide a very effective way of handling complex problems and are good substitutes for existing techniques that normally get trapped in local optima [33]. In addition, most of these algorithms have the capacity to escape from local optima which provides the ability to locate near-optimal solutions in a reasonable amount of time. Also, most metaheuristics are simple to understand, design and implement. In Table 2, some of the specific advantages and disadvantages of the twelve metaheuristic algorithms presented in this paper are outlined. Also, Table 3 shows the motivations for all the presented algorithms.

3 Experimental methodology

This section presents the experimental procedures of the comparative study presented in this paper. There are many significant evaluation methods that have been employed by different researchers to measure the robustness and performance merit of several proposed metaheuristic algorithms. Some of these performance measures were identified by the research conducted in [27]. The current study, however, follows similar evaluation approach by adopting three of the measures of merit discussed in [27] to evaluate the twelve representative algorithms presented in this paper. These performance measures of merit are highlighted as follows:

1. Algorithm scope of applicability with respect to robustness in solving very complex problems with large graph size and high dimensionality.
2. Algorithm run-time or computational complexity, which in this case is used to determine the performance efficiency of the individual algorithm or its order of convergence.
3. Reliability to achieve and maintain a certain level of consistency in terms of success ratio between different competing algorithms.

In addition, the required number of function evaluations for acquiring global minimum for the test problems was also used as a method of evaluation for the representative algorithms. The number of function evaluations (NFE) was chosen in place of the number of iterations due to some ambiguity that is associated with the later. For example, various algorithms associate different meanings and computational effort with iteration. More so, measures of merit that are based on the number of iterations have been discouraged because of the biased and unfair comparison it presents, which is most likely associated with the discrepancies in the number of functions evaluation consumed by the individual compared algorithm.

In this paper, to maintain some level of consistence and fairness in comparisons among the twelve representative algorithms, the evaluation effort was focused on the quality of solutions obtained by each algorithm. While it is important to compare metaheuristics in terms of simplicity of design and implementation by considering their number of control parameters, one of the most important measures of comparisons involves the quality of solutions produced by the metaheuristic. More so, one of the primary goals of metaheuristic design is to give solutions of good quality within a reasonable execution timeframe that is better than those of

Table 2 Advantages and disadvantages of metaheuristic algorithms

Algorithm	Advantages	Disadvantages
Cuckoo search	CSA has the ability to converge to a true global optimum [57]	CSA produces low classification accuracy [58]
	CSA can handle local and global search. It makes use of Levy flight as a strategy for global search [57]	CSA has low convergence rate [58]
Differential evolution	DE is good at exploration and diversification	DE convergence is not stable [59]
	DE has the capacity to deal with cost functions that are non-differentiable, multimodal and nonlinear	DE easily falls into a regional optimum [59]
	DE can handle cost functions with high computational complexity	DE requires parameter tuning
	DE is easy to use; it requires only few parameters [59]	In DE, using the same parameter setting may not produce the global best solution
	DE has the ability to converge to the global minimum	
Genetic algorithm	GA is easy to implement	In GA, there is no guarantee that the global maxima will be identified. It has high likelihood of getting trapped in the local maxima
	GA has the ability to handle random types of objectives and constraints	GA does not have a standard method for defining a good fitness function. The best solutions majorly depend on a fitness function and hence the fitness function must be very accurate
	GA can be used independently to solve a given problem. It does not depend on other algorithms or heuristics	In GA, premature convergence seldom occurs, thus losing the population diversity
	GA can be used to handle problems whose constraints and objective functions are nonlinear or discontinuous	GA does not have standard termination criteria, neither does it have a standard method for adjusting its parameters
	GA uses simple operators and can be used to solve problems that have high computational complexity, such as the TSP problem [60]	GA can be time consuming, especially for problems with a large number of variables
Particle swarm optimisation	Calculation in PSO is simple [61]	In PSO, all solutions converge prematurely and consequently lose the population diversity
	It is useful in scientific research and in engineering [61]	PSO suffers from partial optimism
Symbiotic organisms search	Operation of SOS does not require any specific parameters	Similar to other population-based algorithms, SOS depends on iterative performance, which can be time consuming
	SOS has a very fast rate of convergence and reduced computational time	
	SOS does not require procreation, but adjusts through interactions between individuals in the population [62]	
	SOS differs from other metaheuristic algorithms by three of its unique operators: parasitism, commensalism and mutualism [62]	
Firefly algorithm	FFA has the ability to automatically divide the population into different groups, hence it is good for diversification	Similar to some metaheuristic algorithms, FFA performance depends on adequate parameter tuning
		Diversification in FFA can lead to reduced speed and reduced convergence rate [63]
		FFA is not very suitable for handling complex problems, because it can be trapped in many local optima [63]
Ant colony optimisation	The construction process for ACO is inherently parallel, as ant builds solutions independently and simultaneously [64]	ACO probability distribution changes with iteration [64]
	Distributed computation in ACO evades premature convergence [64]	Although convergence in ACO is guaranteed, convergence time is undefined [65]
	ACO can be used to efficiently handle Travelling Salesman Problem and related problems [64]	It is difficult to theoretically analyse the behaviour of ACO, since ACO is based on sequences of random decisions of different independent artificial ants [64]
	PSO has the ability to adapt to changes, hence it is suitable for dynamic applications [64]	

Table 2 (continued)

Algorithm	Advantages	Disadvantages
Bat algorithm	BA uses frequency tuning to increase the population diversity [43]	The standard BA supports continuous values, hence it can be easily applied to nonlinear global optimisation problems. However, BA cannot efficiently handle discrete optimisation problems [66], hence the need for improved implementations, such as the implementation in [67]. In addition, BA requires several parameters that require tuning [68]
	BA has the capacity to automatically zoom into areas with good solutions. Automatic zooming helps to balance exploration and exploitation during search [43]	
	BA does not use a fixed parameter, but uses parameter control to vary its parameter values at different iterations. This gives BA the ability to automatically and quickly switch from exploration stage to exploitation stage [43]	
	BA is simple, flexible and easy to implement. It can be used to handle a vast number of problems [43]	
Flower pollination algorithm	FPA is flexible, simple, easy to implement, has few parameters and can be used to handle both single and multiple objective optimisation problems [69, 70]	FPA has slow convergence rate and low precision [70]
	Flower constancy in FPA may have evolutionary advantages, because the algorithm maximises the transfer of pollen grain to plants of the same species, which consequently maximises or enhances the production of the same flower species for evolutionary speciation	FPA easily falls into a local optimum [70]
Artificial bee colony algorithm	ABC algorithm is very robust, it converges fast, it requires few parameters and it is flexible [71]	ABC algorithm has premature convergence in the later stage of its search and the classification accuracy of its best obtained value may not be high enough to meet the requirements
The bee algorithm	The BA is easy to implement and it has the ability to perform both local and global search	The BA algorithm requires manual parameter settings and it has several parameters for tuning
	It can be combined with other algorithms	
Invasive weed optimisation	In IWO, all the candidates in the solution space are involved in the reproduction stage, whereas some algorithms (such as GA) do not allow individuals with less fitness value to reproduce [72]	IWO is not very effective for problems with large search space. This is because, a larger number of seeds will be applied to each plant so that the search space can be totally examined. This can lead to an increase in computation time [72]
	The computational cost of IWO is not high [72]	

the exact approaches [34]. The measures of merit that were considered for solution quality in this case was the deviation of solutions returned by the representative algorithms from optimality. However, to avoid giving preference to problem instances with larger optimal solution values, the average percentage error over all test problems was employed, as this strategy gives equal weight to each problem instance.

Three different lengths of algorithm execution were considered, which spanned from 100,000, 300,000, and 500,000 number of function evaluations. The above range of function evaluation (FE) were chosen considering the fact that the difficulty of solving a test problem is affected by its dimensionality, since test problems with a higher dimension would be more difficult to solve than those with smaller dimensions. After several experimental trials, the source of data gathering for the proposed numerical comparison of the twelve algorithms was done. This was achieved by recording the various objective function values obtained by each representative algorithm based on the above-mentioned measures of merit. Descriptive statistics, performance graphs and inferential statistical analyses tests were used to present comprehensive reports on the experimental findings. Details of the experimental configuration, algorithmic parameter settings and discussion of results are presented in the next sections.

4 Experimentation

All of the twelve algorithms are implemented in MATLAB version 8.5.0.197613 (R2015a) and executed on Windows 7 OS, Intel Celeron CPU@3.40 GHz with 3.88 GB of RAM. In evaluating the performance of any metaheuristic algorithm, it is important to note that experimental settings

Table 3 Representative metaheuristic algorithms and their design motivations

S/n	Algorithm	Motivation
1	Genetic algorithm	Process of natural selection
2	Differential evolution	Evolution
3	Particle swarm optimisation	Schooling and flocking patterns of birds and fishes
4	Symbiotic organisms search	Interactive behaviour between different natural organisms
5	Cuckoo search	Parasites of cuckoo bird species and Levy flight behaviour of some bird species and fruit flies
6	Artificial bee colony algorithm	Intelligent foraging behaviour of honey bees
7	Flower pollination algorithm	Pollination process of flowering plants
8	Ant colony optimisation	Foraging behaviour of specific ant species
9	Bat algorithm	Echolocation behaviour of bats
10	Firefly algorithm	Firefly flashlight attribute
11	The bees algorithm	Foraging behaviour of real life honeybee colonies
12	Invasive weed optimisation	Aggressive growth and colonisation of weed plants

are very significant as these can strongly influence the final results obtained from such a particular experiment. Therefore, in order to avoid conflicting results, especially with multiple comparisons like the current study, it is important to carefully set the value of the algorithm's parameter to its optimal value following the procedures discussed in [35, 36]. However, in the case of the current study, the parameters for each algorithm were not optimised, but rather, the associated reference parameters, that were used to evaluate the algorithms by their inventors, are adopted and presented in Table 4. Each algorithm was assigned a population size of 50, and each function is evaluated using the following dimensions: 2, 4, 5, 10, and 30. As mentioned above, a function evaluation limit of 100,000; 300,000; and 500,000 were set as the stopping criteria for all the twelve representative algorithms.

To test the performance of the representative algorithms, 36 mathematical functions have been chosen, which are denoted in this study as F1–F36. Each of the functions have varying characteristics as aforementioned and presented in Table 1 in the introductory section of this paper. The global minimum values obtained for each of the test problems by the representative algorithms are computed from 30 independent replications. The values of experimental data recorded includes global minimum of function evaluation and algorithm execution run-time. The additional information gathered from the conducted simulation results include the computation of average global minimum values "Min", standard deviation of the global minimum "SD" and the best obtained solution "BestOpt" obtained by each algorithm for the 30 replications. The various results obtained for the three sets of number of function evaluations (100,000; 300,000 and 500,000) are shown in Tables 5, 6, 7, 8, 9, 10, 11 and 12. Subsequently, a multiple comparison test was conducted using the global minimum values to determine the best performing algorithm among the twelve representative algorithms.

The Friedman's nonparametric test with $\alpha = 0.05$ and the Wilcoxon signed-rank tests as Post Hoc Tests were used to compare the statistical significant difference among the twelve algorithms. The Friedman's test statistics can only compare the mean ranks between the competing algorithms and accordingly indicates how the algorithms differ, but the test does not distinctively tell where and at what point the difference exists. Therefore, to examine where the actual differences occur, the Wilcoxon signed-rank tests were separately conducted on the different combinations of the competing algorithms with Bonferroni adjustment. The results of the statistical analyses are reported in Tables 13 and 14.

4.1 Results and discussion

Different experiments are carried out to evaluate the performance of the competing algorithms, and the results obtained are presented in Tables 5, 6 and 7. The tables report the average global minimum values and the standard deviations obtained by the representative algorithms for the 36 benchmark functions. The results show that DE generally outperforms the other algorithms for all the test problems. Furthermore, a similar interpretation of the results showed that six (6) out of the twelve (12) algorithms namely DE, SOS, PSO, GA, CS and ACO performed fairly better than the other algorithms. In addition, it is also observed that the quality of solutions produced by each of DE, SOS, CS and ACO improved consistently with an increase in the number of function evaluation. However, PSO and GA seem to require a smaller number of function evaluations to reach optimum solution. Figure 1 illustrates the evolution of mean best error curve for some selected test problems. The best run error, which is defined as the

Table 4 Parameter settings of the algorithms

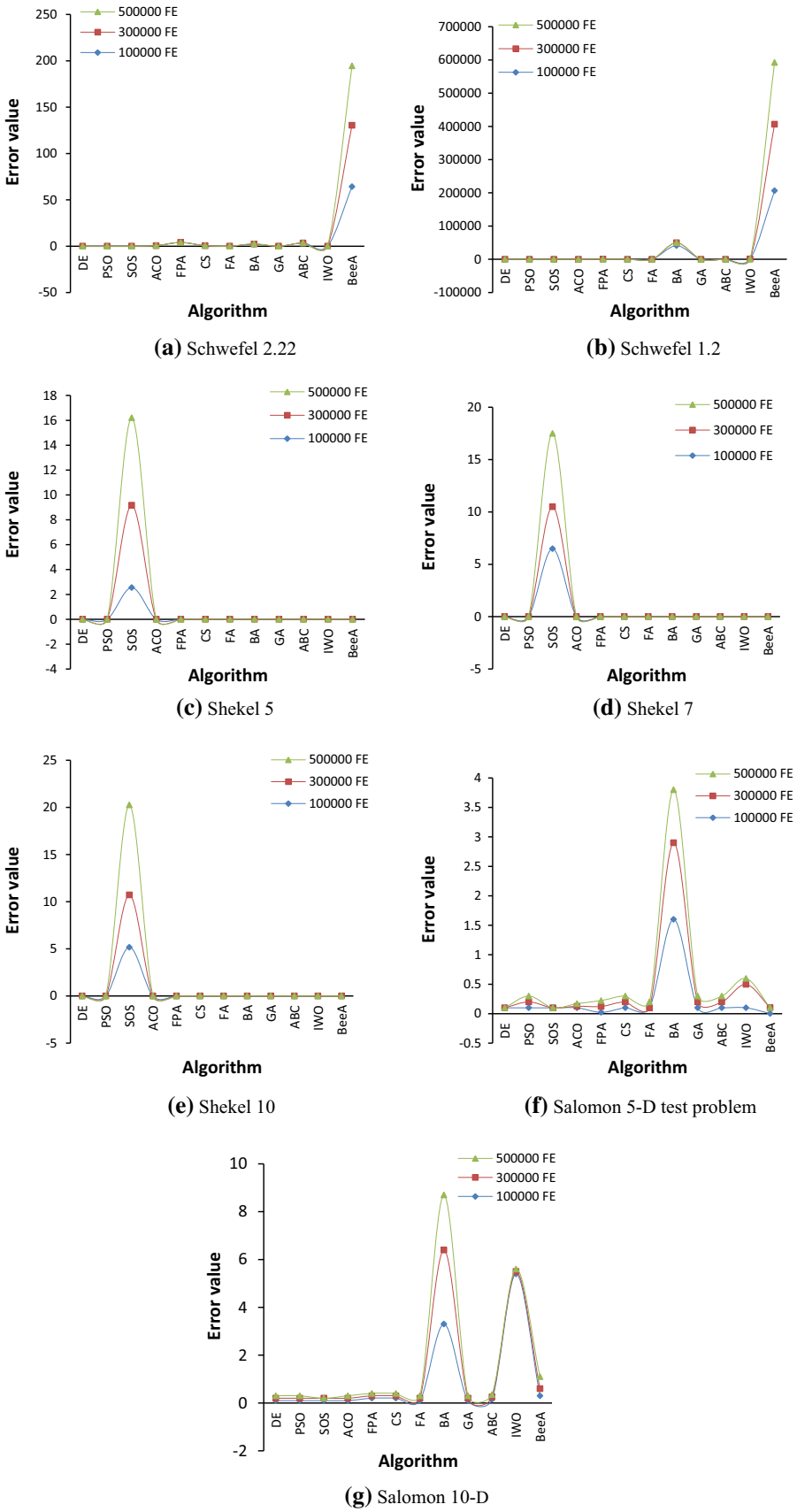
GA	DE	PSO	ACO	FPA	CS	FA	BA	SOS	ABC	IWO	BeeA
$n = 50$	$n = 50$	$n = 50$	$n = 50$	$n = 50$	$n = 50$	$n = 50$	$n = 50$	$n = 50$	$n = 50$	$n = 50$	$n = 50$
$m = 0.01$	$c = 0.9$	$w = 0.99$	$s = 40$	$p = 0.8$	$p_a = 0.25$	$\gamma = 1$	$n_g = 2000$, 6000, 10,000		nOnlooker = n	$s = 5$	e = NP/2
$c = 0.8$	$f_1 = 0.2$	$v = X_{min}/10 \sim X_{max}/10$	$q = 0.5$		$\alpha = 1$	$\beta = 0.2$	$A = 0.5$		$L = r$ (0.6*D*n)	$\Sigma_1 = 0.5$	b = NP/4
$g = 0.9$	$f_2 = 0.8$	$c_1 = 1.5$	$\zeta =$			$\alpha_1 = 0.25$	$r = 0.5$		$a = 1$	$\Sigma_2 = 0.001$	r = NP/4
		$c_2 = 2$				$\alpha_2 = 0.98$	$\lambda = 2$			$e = 2$	$n_1 = 2$
						$m = 2$					$n_2 = 1$

n population size/colony size/ecosystem size, m mutation rate, γ light absorption coefficient, β attraction coefficient base value, α_1 randomness, c crossover rate, f_1 lower bound of scaling factor, f_2 upper bound of scaling factor α_2 mutation coefficient damping ratio, δ uniform mutation range, D number of decision variables, L Stagnation limit, a acceleration coefficient, c_1 personal learning coefficient, c_2 global learning coefficient, ζ deviation-distance ratio, q intensification factor, s sample size, p probability switch, p_a rate of alien eggs or solutions discovery, A loudness, r pulse rate, λ frequency, e variance reduction exponent, Σ_1 initial value of standard deviation, Σ_2 final value of standard deviation

absolute difference between the best run value found by each algorithm denoted by $f(x^*)$ and the actual function optimum value denoted by $f(x')$, mathematically, this is represented by the expression error $= |f(x^*) - f(x')|$. Subsequently, comparing the efficiency of the competing sets of algorithms from the same table, SOS produced the least average CPU time of 1.99 s (100, 000 FE), 4.64 s (300, 000 FE) and 5.93 s (500,000 FE) for the different problem dimensions. On the other hand, DE, PSO, ACO and FA in ascending order produced the highest average CPU time on the same benchmark problems. The time complexity behaviour of each of the twelve methods on the 36 test problems for varying number of function evaluations is illustrated in Figs. 2, 3 and 4.

Tables 8, 9 and 10 show the best results obtained by each of the methods, while in Tables 11 and 12, a summary of the global minimum found by each competing algorithm for the 36 benchmark functions are reported. Specifically, Table 11 shows the number of functions where each algorithm was able to reach the global minimum. To further support and confirm the initial claims made earlier from the results presented in Tables 5, 6 and 7, the results presented in Table 13 show that the best performed algorithms according to the computed percentage success ratio metric are the DE, PSO and GA methods, which in each case found 11% (100,000 FE), 10% (300,000 FE) and 10% (500,000 FE) of the global minimum solution of the 36 test problems for the 30 replication runs, respectively. The SOS algorithm closely follows the three best algorithms with percentage success ratios of 10% (100,000 FE), 10% (300,000 FE) and 9% (500,000 FE), respectively. The ABC and FA algorithms were the least performed algorithms with minimum percentage success ratios of 6% (100,000), 5% (300,000 FE) and 7% (500,000 FE), respectively. It is worth indicating that out of the twelve representative algorithms, only the SOS algorithm was able to find a global minimum for the benchmark test problems F35 and F36. Therefore, this shows that the algorithm might have some degree of latent performance potential that is worth further investigation by interested optimisation researchers.

In Tables 11 and 12, the performance of each of the algorithms with respect to the predefined maximum numbers of function evaluations are summarised. Clearly, the DE, PSO, GA and SOS illustrate promising results by dominating the other algorithms throughout the experimental process. Although, when the maximum numbers of function evaluations are 300,000 and 500,000, the ACO and CS also performed moderately well as compared to the other remaining algorithms. However, the IWO, FA, BA, ABC, BeeA, and FPA appeared to be the least performed algorithms. In addition, based on the computed percentage success ratio for each of the twelve algorithms, it is very clear that the DE, PSO, GA, and SOS still maintained a



(a) Schwefel 2.22

(b) Schwefel 1.2

(c) Shekel 5

(d) Shekel 7

(e) Shekel 10

(f) Salomon 5-D test problem

(g) Salomon 10-D

Fig. 1 Mean best error curve of twelve algorithms using a different number of function evaluations for selected test problems

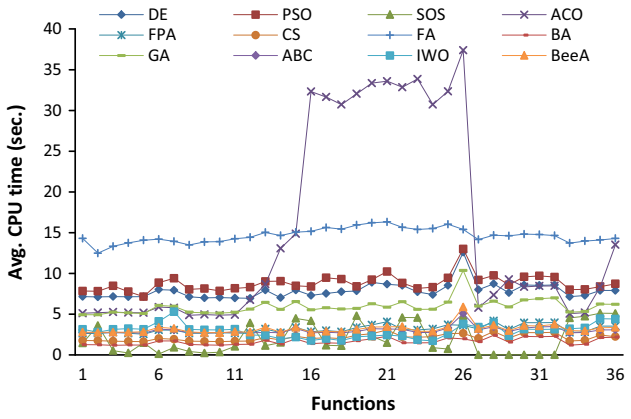


Fig. 2 Average CPU time in seconds for running each of the twelve algorithms for 100,000 FE

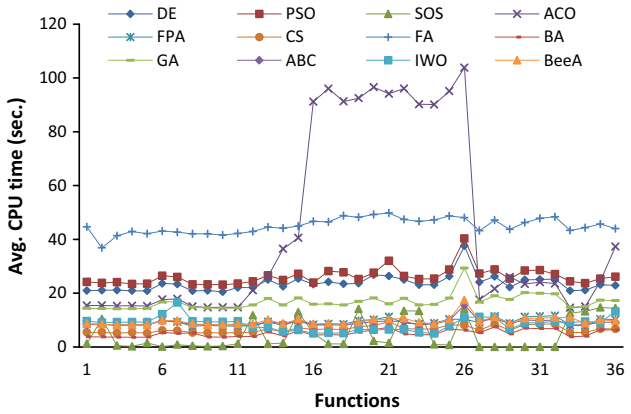


Fig. 3 Average CPU time in seconds for running each of the twelve algorithms for 300,000 FE

lead in the overall performance evaluation. Therefore, the best algorithms to recommend to practitioners who are willing to use the three maximum numbers of function evaluations used in this paper, are the DE, PSO, GA, and SOS.

Figure 1 illustrates the effect that a stopping criterion, which is defined in this paper as the maximum number of function evaluations, has on the performance of the individual algorithms. As mentioned earlier, three different levels of maximum numbers of function evaluations were used to evaluate the representative algorithms. In Fig. 1a–g, the performance history of mean best error curve for all the tested algorithms, for each case of 100,000, 300,000, and 500,000 FE are shown, respectively. The different subgraphs in Fig. 1 clearly show that some of the algorithms demonstrated better performances with larger number of function evaluations such as the DE, ACO, CS, FPA, ABC, and BeeA, while some generated good quality results with small numbers of function evaluations such as the IWO and BA. Furthermore, in some instances, a change

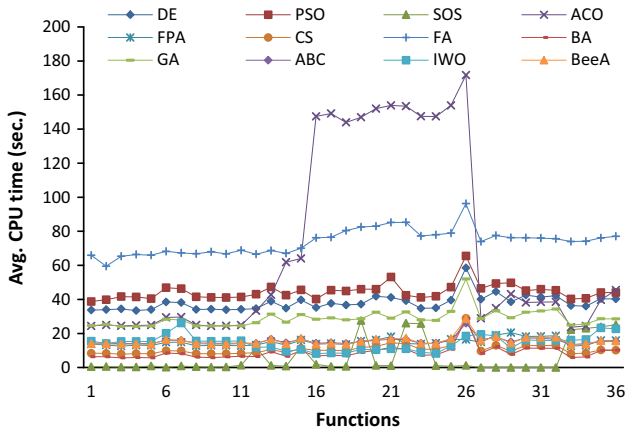


Fig. 4 Average CPU time in seconds for running each of the twelve algorithms for 500,000 FE

in the number of function evaluation did not affect the performance of the algorithm, as it is the case with PSO and GA. In Fig. 1c–e, it can be seen that the SOS is outperformed by all the other algorithms on the problem instances namely, Shekel 5, Shekel 7 and Shekel 10, whereas, according to the evaluation based on the computed percentage success ratios, the FA and ABC were the least performed algorithms. However, the overall evaluation revealed that the DE, PSO, GA, and SOS were the best performed algorithms, while ACO and CS performed moderately well.

The comparison based on the individual algorithm execution time complexity were only considered here because all the algorithms were implemented under the same experimental condition, namely, using the same programming language, computer system, and parameter settings. Figures 2, 3 and 4 show the performance of all the algorithms in terms of their average CPU execution time. The SOS outperformed all the other algorithms by having the least average CPU time for the different problem dimensions and maximum number of function evaluations. On the other hand, even though the DE and PSO produced better solutions, their respective average CPU time were quite high on the same benchmark problems. However, the FA was outperformed by all the other algorithms.

4.2 Statistical test

The results of the descriptive statistical analyses are presented above, which include the mean of the global minimum, success ratio, standard deviation and best global minimum solution found by the various representative algorithms. In addition, the results of the Friedman's test for multiple comparison of the twelve algorithms performances on the 36 benchmark functions are tabulated in Tables 13 and 14 (an experiment involves 30 trials as

Table 5 The average global minimum values of the twelve algorithms for 100,000 function evaluation

No.	Functions	DE	PSO	SOS	ACO	FPA	CS	FA	BA	GA	ABC	IWO	BeeA
1	F1												
	Min	0	0	0	0	0	0	1.74E−10	0.125134	2.45E−05	7.39E−08	1.44E−09	1.92E−05
	SD	0	0	0	0	0	0	1.5E−10	0.233148	6.56E−05	8.67E−08	1.48E−09	1.37E−05
2	F2												
	Min	− 1	− 1	− 1	− 1	− 1	− 1	− 0.83333	− 0.2667	− 1	− 1	− 0.53333	− 1
	SD	0	0	0	0	0	0	0.379049	0.449753	0	0	0.507416	0
3	F3												
	Min	0	0	0	0	0	0	4.95E−11	0	3E−08	1.28E−09	4.25E−11	7.04E−07
	SD	0	0	0	0	0	0	4.09E−11	0	8.53E−08	1.37E−09	3.56E−11	5.52E−07
4	F4												
	Min	0	0	0	0	0	0	1.08E−06	2.612191	0	1.19E−09	8.91E−09	3.01E−06
	SD	0	0	0	0	0	0	1.26E−06	1.209386	0	1.54E−09	8.99E−09	2.41E−06
5	F5												
	Min	0.118356	0.118356	0	0.373265	0	0	0.016907	0.287436	0.237458	0.009288	0.16908	1.28E−07
	SD	0.218206	0.218206	0	0.196866	0	0	0.092602	0.255652	0.258202	0.008884	0.243203	1.56E−07
6	F6												
	Min	− 1.8013	− 1.8013	− 1.8013	− 1.8013	− 1.8013	− 1.8013	− 1.8013	− 1.8013	− 1.8013	− 1.98375	− 1.8013	− 1.8013
	SD	0	0	0	0	0	0	0	0	0	0.012897	0	0
7	F7												
	Min	0	0	0	0	6.09E−11	8.41E−09	0.003685	0.246737	0.01019	0.001306	0.433195	6.52E−07
	SD	0	0	0	0	1.67E−10	4.6E−08	0.00407	0.181949	0.018787	0.001405	0.125207	7.61E−07
8	F8												
	Min	− 1.0316	− 1.0316	− 1.0316	− 1.0316	− 1.0316	− 1.0316	− 1.0316	− 1.0316	− 1.0316	− 1.0316	− 1.0316	− 1.0316
	SD	0	0	0	0	0	0	0	0	0	0	0	0
9	F9												
	Min	0	0	0	0	0	0	4.34E−08	14.79769	0	5.19E−11	9.54E−10	1.73E−07
	SD	0	0	0	0	0	0	4.26E−08	24.90513	0	6.02E−11	1.13E−09	2.03E−07
10	F10												
	Min	0	0	0	0	0	0	4.05E−07	1.668928	7.79E−05	3.41E−07	3.65E−09	0.000195
	SD	0	0	0	0	0	0	4.21E−07	1.467871	0.00034	5.85E−07	3.35E−09	0.000176
11	F11												
	Min	− 186.731	− 186.731	− 186.731	− 186.56	− 186.731	− 186.731	− 186.731	− 186.731	− 186.731	− 186.525	− 186.731	− 186.731
	SD	0	0	0	0.163398	0	0	0	0	0	0.205154	0	0
12	F12												
	Min	0.002084	3.55E−07	1.7E−06	0.003802	0.00012	2.73E−08	1.31E−05	0.253928	0.389614	0.057231	6.24E−06	0.073689
	SD	0.007996	5.84E−07	3.13E−06	0.003737	0.000484	6.75E−08	2.58E−05	1.389662	0.366744	0.045029	3.58E−06	0.11937

Table 5 (continued)

No.	Functions	DE	PSO	SOS	ACO	FPA	CS	FA	BA	GA	ABC	IWO	BeeA
13	F13												
	Min	− 4.68631	− 4.53059	− 4.6877	− 4.61488	− 4.63889	− 4.6877	− 4.6459	− 3.55343	− 4.6877	− 4.62584	− 4.6459	− 4.6254
	SD	0.007632	0.141476	0	0.232736	0.028424	0	0	0.532113	0	0.101688	0	0.06341
14	F14												
	Min	6.07E−11	0	0	0	3.55E−07	2.42E−08	9.62E−07	1.37E−06	0	13.21387	4.61E−06	5.271545
	SD	9.64E−11	0	0	0	2.14E−07	1.55E−08	3.49E−07	2.78E−07	0	5.0191	1.34E−06	4.295504
15	F15												
	Min	− 9.6602	− 8.89867	− 9.52815	− 5.42275	− 7.73548	− 8.77987	− 8.70307	− 5.85612	− 9.65878	− 6.1317	− 8.972	− 8.60627
	SD	0	0.454675	0.143799	0.609768	0.340891	0.227361	0.475663	0.888662	0.007627	0.35998	0.312243	0.34362
16	F16												
	Min	0	0	0	2.28E−05	0.127591	2.68E−05	3.15E−06	1.3E−05	2.98E−08	0.023309	4.53E−05	64.19777
	SD	0	0	0	1.19E−05	0.093202	1.29E−05	7.57E−07	1.67E−06	9.2E−08	0.006879	6.9E−06	9.728679
17	F17												
	Min	0	0	0	0.008633	39.37695	0.008857	0.001294	9125.711	9.46E−06	9.144857	379.2095	24,313.46
	SD	0	0	0	0.004304	22.26435	0.003347	0.000304	2361.953	3.17E−05	2.848034	367.5795	3740.928
18	F18												
	Min	0	0	0	0.001204	4.205013	0.001122	0.00812	0.000785	6.12E−07	1.163459	0.001275	2868.832
	SD	0	0	0	0.00058	2.590954	0.000375	0.009599	0.002265	2.53E−06	0.471607	0.000671	561.1831
19	F19												
	Min	0.011499	0.005325	0.000478	0.101121	0.050035	0.035653	0.022093	1.786923	0.000557	0.331196	0.019056	15.15221
	SD	0.002165	0.001946	0.000222	0.026612	0.016122	0.012944	0.020519	0.574149	0.000319	0.05801	0.00663	4.003753
20	F20												
	Min	0	0.005464	0	37.53509	7.58112	1.428207	0.042577	6309.875	1.14E−06	30.36138	0.031171	81.26146
	SD	0	0.017385	0	30.31922	1.917772	1.013607	0.022516	34,191.24	6.22E−06	20.04855	0.002391	6.917832
21	F21												
	Min	0	0	0	0.11226	441.1606	0.11673	0.07066	150031	2.58E−05	124.834	12,240.02	292,753.9
	SD	0	0	0	0.079245	202.6607	0.038771	0.054372	63,077.71	8.89E−05	41.72095	6734.093	39,514.47
22	F22												
	Min	34.26736	26.04939	18.15055	3133.414	1102.279	40.51227	57.14667	2.750572	7.034444	314,624.7	46.32691	25,215,399
	SD	17.08378	22.67168	1.141894	2722.962	531.4646	10.90828	73.92022	7.940751	11.7931	154,961.5	44.98647	7,794,324
23	F23												
	Min	0.66667	0.66667	0.66667	13.25126	12.97718	0.74858	0.926493	0.675533	0.571138	1633.865	0.730624	159,639.4
	SD	0	0	0	6.730554	8.991301	0.066418	0.483475	0.036639	0.299638	853.3791	0.087002	47,631.7
24	F24												
	Min	37.76777	46.16602	0	241.0779	89.34264	87.13682	31.97192	84.60685	2.23E−05	228.9718	63.6866	144.9444
	SD	4.400108	11.62541	0	16.92551	14.29723	12.38445	11.01365	34.12127	5.14E−05	16.55505	19.40744	13.63913

Table 5 (continued)

No.	Functions	DE	PSO	SOS	ACO	FPA	CS	FA	BA	GA	ABC	IWO	BeeA
25	F25												
	Min	0	0.02721	0	0.194886	1.312787	0.09083	0.002713	152.0818	0.019158	1.071133	405.4714	237.4175
	SD	0	0.027512	0	0.257887	0.158625	0.045723	0.000684	57.59136	0.0139	0.022867	54.50678	31.76004
26	F26												
	Min	0	0.979077	0	0.027602	2.98876	4.14612	0.00839	15.51748	0.000382	3.497323	18.07643	18.25756
	SD	0	0.723107	0	0.014651	0.467586	1.265781	0.000829	1.172301	0.000751	0.585704	3.472498	0.327028
27	F27												
	Min	− 1	− 1	− 0.90094	− 1	− 1	− 1	− 1	− 0.90828	− 0.9915	− 0.99988	− 1	− 1
	SD	0	0	0.043076	0	0	0	0	0.063378	0.022041	0.000147	0	0
28	F28												
	Min	− 3.8628	− 3.8628	− 3.74152	− 3.8628	− 3.8628	− 3.8628	− 3.8628	− 3.8628	− 3.8628	− 3.8628	− 3.8628	− 3.8628
	SD	0	0	0.083809	0	0	0	0	0	0	0	0	0
29	F29												
	Min	− 3.3224	− 3.3224	− 2.68766	− 3.3224	− 3.3224	− 3.3224	− 3.3224	− 3.2032	− 3.28664	− 3.3224	− 3.2032	− 3.3224
	SD	0	0	0.268865	0	0	0	0	0	0.055558	0	0	0
30	F30												
	Min	− 10.1532	− 5.97419	− 1.38021	− 5.15602	− 10.1532	− 10.1532	− 10.1532	− 4.89603	− 7.40338	− 10.0694	− 5.55828	− 10.1532
	SD	0	3.371298	1.253714	3.529503	0	0	0	3.098144	3.491636	0.437906	3.221693	0
31	F31												
	Min	− 10.4029	− 7.91495	− 1.66077	− 8.41198	− 10.4029	− 10.4029	− 10.4029	− 6.10001	− 8.17576	− 10.4029	− 7.61745	− 10.1532
	SD	0	3.398333	0.750023	3.00258	0	0	0	3.457059	3.476117	0	3.11688	0
32	F32												
	Min	− 10.5364	− 7.66899	− 1.67744	− 9.33238	− 10.5364	− 10.5364	− 10.5364	− 5.70467	− 8.43545	− 10.5364	− 7.03466	− 10.4029
	SD	0	3.841268	0.819012	2.754959	0	0	0	3.610903	3.30999	0	3.864091	0
33	F33												
	Min	0.39789	0.39789	0.39789	0.39789	0.39789	0.3979	0.3979	0.39789	0.39789	0.39789	0.39789	0.39789
	SD	0	0	0	0	0	0	0	0	0	0	0	0
34	F34												
	Min	3	3	3	3	3	3	3	3	3	3	3	3
	SD	0	0	0	0	0	0	0	0	0	0	0	0
35	F35												
	Min	0.099873	0.099873	0.099873	0.099873	0.097303	0.0999	0.0999	3.233233	0.106539	0.099886	1.716557	0.100256
	SD	0	0	0	0	0.014075	0	0	1.228493	0.02537	2.29E−05	1.104573	0.044621
36	F36												
	Min	0.099873	0.099873	0.099873	0.099873	0.280677	0.1999	0.0999	5.6599	0.183204	0.207005	7.653233	2.146565
	SD	0	0	0	0	0.048644	0	0	1.35611	0.069892	0.027902	1.616026	0.857721
	Average CPU time	7.854801	8.796963	1.991839	15.03402	3.151385	2.125924	14.6453	1.63661	5.934088	2.997953	2.930405	3.17859

Table 6 The average global minimum values of the twelve algorithms for 300,000 function evaluation

No.	Functions	DE	PSO	SOS	ACO	FPA	CS	FA	BA	GA	ABC	IWO	BeeA
1	F1												
	Min	0	0	0	0	0	0	7.5E−11	0.177399	3.87E−06	1.93E−08	3E−10	4.53E−11
	SD	0	0	0	0	0	0	1.14E−10	0.257151	7.45E−06	1.99E−08	3.15E−10	1.72E−11
2	F2												
	Min	− 1	− 1	− 1	− 1	− 1	− 1	− 0.63333	− 0.1667	− 1	− 1	− 0.93333	− 1
	SD	0	0	0	0	0	0	0.490133	0.379032	0	0	0.253708	0
3	F3												
	Min	0	0	0	0	0	0	0	0	7.24E−08	2.8E−10	2.32E−11	0
	SD	0	0	0	0	0	0	1.66E−11	0	3.07E−07	3.42E−10	2.32E−11	0
4	F4												
	Min	0	0	0	0	0	0	2.13E−07	2.60042	0	3.51E−10	6.33E−09	0
	SD	0	0	0	0	0	0	1.7E−07	0.979315	0	3.46E−10	4.71E−09	0
5	F5												
	Min	0	0.4227	0	0.343503	0	0	2.82E−09	0.287436	0.152587	0.004302	0.202896	0
	SD	0	0.192269	0	0.214024	0	0	3.03E−09	0.255652	0.237066	0.007227	0.252744	0
6	F6												
	Min	− 1.8013	− 1.8013	− 1.8013	− 1.8013	− 1.8013	− 1.8013	− 1.8013	− 1.85496	− 1.8013	− 1.99601	− 1.8013	− 1.8013
	SD	0	0	0	0	0	0	0	0.083908	0	0.003393	0	0
7	F7												
	Min	0	0	0	0	0	0	0.001587	0.233757	0.018924	0.000493	0.43397	0
	SD	0	0	0	0	0	0	0.002098	0.184708	0.02201	0.000499	0.141285	0
8	F8												
	Min	− 1.0316	− 1.0316	− 1.0316	− 1.0316	− 1.0316	− 1.0316	− 1.0316	0.036393	− 1.0316	− 1.0316	− 1.0316	− 1.0316
	SD	0	0	0	0	0	0	0	0.016553	0	0	0	0
9	F9												
	Min	0	0	0	0	0	0	2.64E−08	12.80358	0	0	3.09E−10	0
	SD	0	0	0	0	0	0	3.11E−08	20.85816	0	0	2.27E−10	0
10	F10												
	Min	0	0	0	0	0	0	6.71E−08	1.646757	4.94E−05	1.1E−07	1.39E−09	2.63E−09
	SD	0	0	0	0	0	0	7.12E−08	1.367106	0.000127	1.19E−07	1.9E−09	1.6E−09
11	F11												
	Min	− 186.731	− 186.731	− 186.73	− 186.688	− 186.731	0	− 186.731	− 186.731	− 186.731	− 186.623	− 186.731	− 186.731
	SD	0	0	0	0.038565	0	0	0	0	0	0.079705	0	0
12	F12												
	Min	3.66E−06	0	0	1.65E−06	0	0	1.55E−06	2.97E−07	0.377487	0.011674	2.43E−06	0.0362
	SD	1.99E−05	0	0	1.57E−06	0	0	1.05E−06	1.75E−07	0.245328	0.010595	1.22E−06	0.101628

Table 6 (continued)

No.	Functions	DE	PSO	SOS	ACO	FPA	CS	FA	BA	GA	ABC	IWO	BeeA
13	F13												
	Min	− 4.6877	− 4.5377	− 4.6877	− 4.64941	− 4.6877	− 4.6877	− 3.6659	− 3.54069	− 4.6877	− 4.60131	− 4.64704	− 4.64637
	SD	0	0	0	0.171731	0	0	0.061206	0.536955	0	0.097263	0.045072	0.053018
14	F14												
	Min	0	0	0	0	0	0	9.58E−07	8.83E−07	0.010732	4.747253	2.82E−06	4.227042
	SD	0	0	0	0	0	0	4.0183E−07	1.97E−07	0.005171	2.28803	7.92E−07	3.7151
15	F15												
	Min	− 9.6602	− 8.97165	− 9.6565	− 5.72682	− 8.56919	− 9.4317	− 8.85992	− 5.93769	− 9.6602	− 5.94075	− 9.06729	− 8.45413
	SD	0	0.43584	0.009482	0.650794	0.262538	0.075079	0.365879	1.060252	0	0.327318	0.24896	0.362679
16	F16												
	Min	0	0	0	0	9.99E−06	0	1.71E−06	1.06E−05	1.94E−09	1.31E−07	2.82E−05	66.36413
	SD	0	0	0	0	1.32E−05	0	3.52E−07	1.05E−06	5.52E−09	1.47E−07	4.2E−06	7.264785
17	F17												
	Min	0	0	0	0	0.002514	0	0.00061	3889.279	3.9E−07	4.54E−05	2.8E−05	23,943.82
	SD	0	0	0	0	0.00291	0	0.000104	2155.015	1.49E−06	4.97E−05	3.69E−06	4512.327
18	F18												
	Min	0	0	0	0	0.000332	0	0.000359	0.000185	1.5E−06	6.56E−06	0.000441	2883.337
	SD	0	0	0	0	0.000436	0	0.000219	3.34E−05	5.48E−06	6.91E−06	5.44E−05	433.6625
19	F19												
	Min	0.003564	0.002083	0.000155	0.025673	0.015663	0.011037	0.019605	0.680539	0.000199	0.11738	0.006301	15.04062
	SD	0.001035	0.000977	5.91E−05	0.008558	0.006167	0.002656	0.019617	0.208213	0.000118	0.032374	0.001887	3.013147
20	F20												
	Min	0	0.007818	0	1.75E−08	0.07169	3.39E−06	0.014013	37.46702	2.74E−06	0.003923	0.023799	78.79434
	SD	0	0.016099	0	5.78E−08	0.023314	1.98E−06	0.002774	66.34511	8.07E−06	0.008927	0.001771	6.167008
21	F21												
	Min	0	0	0	0	0.021765	0	0.0215	62,284.76	1.7E−05	0.000652	0.00043	269,619.8
	SD	0	0	0	0	0.020924	0	0.011859	38,517.35	5.07E−05	0.001442	7.02E−05	37,121.63
22	F22												
	Min	26.18462	31.15628	6.792537	24.81089	29.7541	16.66999	38.86576	34.52658	4.559312	9564.628	84.78078	23,995,537
	SD	9.585736	24.47534	2.210108	0.201134	15.29949	3.192499	34.31393	61.35202	3.56795	8879.945	111.3976	7,084,361
23	F23												
	Min	0.66667	0.66667	0.66667	0.66667	0.679622	0.6667	0.70166	0.667077	0.440574	9.032073	0.678643	149,779.5
	SD	0	0	0	0	0.044512	0	0.066705	0.000236	0.340876	3.151557	0.029225	45,510.01
24	F24												
	Min	0	25.37472	0	216.2763	58.79262	50.7355	35.02287	83.14718	2.33E−06	216.4403	46.37074	143.5056
	SD	0	0.235326	0	14.33827	10.74184	7.046924	8.544581	40.50341	4.49E−06	11.05757	14.7081	15.1652

Table 6 (continued)

No.	Functions	DE	PSO	SOS	ACO	FPA	CS	FA	BA	GA	ABC	IWO	BeeA
25	F25												
	Min	0	0.013267	0	0.024944	0.064364	1.84E−08	0.002102	112.6522	0.025758	0.091988	155.7569	223.2058
	SD	0	0.017928	0	0.065464	0.055702	6.2E−08	0.002304	31.31716	0.017195	0.117755	33.38263	28.2113
26	F26												
	Min	0	1.087033	0	2.45E−09	0.360849	0.17179	0.005937	15.27117	0.000199	0.15233	14.31097	18.17821
	SD	0	0.813943	0	1.26E−09	0.605641	0.403948	0.000689	1.039413	0.000419	0.139182	8.033034	0.316929
27	F27												
	Min	− 1	− 1	− 0.88458	− 1	− 1	− 1	− 1	− 0.95538	− 1	− 0.99997	− 1	− 1
	SD	0	0	0.060389	0	0	0	0	0.029713	0	3.76E−05	0	0
28	F28												
	Min	− 3.8628	− 3.8628	− 3.76383	− 3.8628	− 3.8628	− 3.8628	− 3.8628	− 3.83703	− 3.8628	− 3.8628	− 3.8628	− 3.8628
	SD	0	0	0.065993	0	0	0	0	0.14113	0	0	0	0
29	F29												
	Min	− 3.3224	− 3.3224	− 2.66912	− 3.28267	− 3.3224	− 3.3224	− 3.28664	− 3.26677	− 3.3224	− 3.3224	− 3.2032	− 3.3224
	SD	0	0	0.241698	0.057152	0	0	0.055558	0.060484	0	0	0	0
30	F30												
	Min	− 10.1532	− 5.8888	− 1.29244	− 10.1532	− 10.1532	− 10.1532	− 9.98479	− 5.13433	− 7.23869	− 10.1532	− 6.55676	− 10.1532
	SD	0	3.423882	0.612783	0	0	0	0.922438	3.008919	3.656347	0	3.522763	0
31	F31												
	Min	− 10.4029	− 7.50239	− 1.75314	− 9.09395	− 10.4029	− 10.4029	− 10.4029	− 5.96828	− 8.58732	− 10.4029	− 7.43446	− 10.4029
	SD	0	3.656702	1.057626	2.698397	0	0	0	3.342292	3.104273	0	3.344894	0
32	F32												
	Min	− 10.5364	− 7.78822	− 1.70654	− 9.74316	− 10.5364	− 10.5364	− 10.5364	− 5.91197	− 9.36999	− 10.5364	− 7.52057	− 10.5364
	SD	0	3.70402	0.822028	2.147481	0	0	0	3.893476	2.687304	0	3.612777	0
33	F33												
	Min	0.39789	0.39789	0.39789	0.39789	0.39789	0.3979	0.3979	0.39789	0.39789	0.39789	0.39789	− 10.5364
	SD	0	0	0	0	0	0	0	0	0	0	0	0
34	F34												
	Min	3	3	3	3	3	3	3	3	3	3	3	3
	SD	0	0	0	0	0	0	0	0	0	0	0	0
35	F35												
	Min	0.068346	0.099873	0.045747	0.087597	0.099873	0.0999	0.086622	3.483233	0.103206	0.099875	0.39789	0.099873
	SD	0.045853	0	0.041628	0.021277	0	0	0.03443	1.152833	0.018257	2.58E−06	0	0
36	F36												
	Min	0.099873	0.173204	0.099873	0.099873	0.099873	0.0999	0.0999	6.153233	0.163205	0.147153	0.50321	2.196564
	SD	0	0.058328	0	0	0	0	0	1.245332	0.061493	0.046681	0.462749	0.830661
Average CPU time		23.60292	26.20076	4.638246	43.33359	9.168057	6.563446	44.96854	5.080318	16.75745	9.07315	8.526317	9.594685

Table 7 The average global minimum values of the twelve algorithms for 500,000 function evaluation

No.	Functions	DE	PSO	SOS	ACO	FPA	CS	FA	BA	GA	ABC	IWO	BeeA
1	F1												
	Mean	0	0	0	0	0	0	3.68E−11	0.105372	2.05E−06	1.01E−08	1.9E−10	0
	SD	0	0	0	0	0	0	3.08E−11	0.215054	3.28E−06	1.09E−08	1.71E−10	0
2	F2												
	Mean	− 1	− 1	− 1	− 1	− 1	− 1	− 1	− 0.33335	− 1	− 1	− 0.83333	− 1
	SD	0	0	0	0	0	0	0	0.479454	0	0	0.379049	0
3	F3												
	Mean	0	0	0	0	0	0	1.06E−11	7.65E−13	5.9E−08	1.62E−10	1.12E−11	0
	SD	0	0	0	0	0	0	1.06E−11	0	1.35E−07	1.49E−10	1.56E−11	0
4	F4												
	Mean	0	0	0	0	0	0	1.92E−07	2.926341	0	2.37E−10	2.74E−09	0
	SD	0	0	0	0	0	0	1.63E−07	0.851604	0	2.46E−10	2.32E−09	0
5	F5												
	Mean	0	0	0	0	0	0	1.3E−09	0.236712	0.2199	0.003738	0.152172	0
	SD	0	0	0	0	0	0	1.39E−09	0.257382	0.255764	0.004903	0.23642	0
6	F6												
	Mean	− 1.8013	− 1.8013	− 1.8013	− 1.8013	− 1.8013	− 1.8013	− 1.8013	− 1.81966	− 1.8013	− 1.9983	− 1.8013	− 1.8013
	SD	0	0	0	0	0	0	0	0.056083	0	0.001377	0	0
7	F7												
	Mean	0	0	0	0	0	0	0.000725	0.229658	0.013101	0.000268	0.457811	0
	SD	0	0	0	0	0	0	0.00061	0.166812	0.020355	0.000284	0.098122	0
8	F8												
	Mean	− 1.0316	− 1.0316	− 1.0316	− 1.0316	− 1.0316	− 1.0316	− 1.0316	− 1.0316	− 1.0316	− 1.0316	− 1.0316	− 1.0316
	SD	0	0	0	0	0	0	0	0	0	0	0	0
9	F9												
	Mean	0	0	0	0	0	0	1.3E−08	14.80925	0	0	2.52E−10	0
	SD	0	0	0	0	0	0	1.04E−08	23.83941	0	0	1.89E−10	0
10	F10												
	Mean	0	0	0	0	0	0	8.61E−08	1.534334	4.4E−06	6.04E−08	6.33E−10	0
	SD	0	0	0	0	0	0	8.05E−08	1.299631	1.46E−05	5.39E−08	6.47E−10	0
11	F11												
	Mean	− 186.731	− 186.731	− 186.73	− 186.706	− 186.731	− 186.731	− 186.731	− 178.31	− 186.731	− 186.683	− 186.731	− 186.731
	SD	0	0	0.000265	0.023844	0	0	0	21.83527	0	0.035997	0	0
12	F12												
	Mean	0	0	0	1.74E−09	0	0	9.36E−07	1.96E−07	0.252108	0.00703	1.74E−06	0.061699
	SD	0	0	0	4.39E−09	0	0	4.67E−07	1.09E−07	0.248563	0.005797	7.58E−07	0.180957

Table 7 (continued)

No.	Functions	DE	PSO	SOS	ACO	FPA	CS	FA	BA	GA	ABC	IWO	BeeA
13	F13												
	Mean	− 4.6877	− 4.51818	− 4.6877	− 4.7187	− 4.6877	− 4.6877	− 4.58519	− 3.73409	− 4.6877	− 4.55603	− 4.6504	− 4.63959
	SD	0	0.17518	0	0.095547	0	0	0.080293	0.536801	0	0.078353	0.04781	0.062518
14	F14												
	Mean	0	0	0	0	0	0	4.15E−07	7.11E−07	0.004059	2.813027	2.38E−06	4.251958
	SD	0	0	0	0	0	0	1.49E−07	1.47E−07	0.001878	1.799098	6.38E−07	3.045064
15	F15												
	Mean	− 9.6602	− 8.8309	− 9.6602	− 6.14073	− 8.89603	− 9.59767	− 8.75486	− 5.90763	− 9.6602	− 6.282	− 9.15156	− 8.51747
	SD	0	0.549215	0	0.618152	0.232132	0.03421	0.458882	0.876196	0	0.452652	0.242466	0.28854
16	F16												
	Mean	0	0	0	0	9.49E−10	0	1.33E−06	9.53E−06	1.66E−09	0	2.37E−05	63.91914
	SD	0	0	0	0	2.78E−09	0	2.07E−07	9.4E−07	3.09E−09	0	2.35E−06	9.779647
17	F17												
	Mean	0	0	0	0	2.29E−07	0	0.00048	951.9305	1.09E−06	3.7E−10	2.49E−05	23,294.87
	SD	0	0	0	0	2.86E−07	0	8.05E−05	1742.523	4.09E−06	3.43E−10	3.05E−06	3290.597
18	F18												
	Mean	0	0	0	0	3.44E−08	0	0.000194	0.000157	1.14E−08	4.37E−11	0.000364	2688.916
	SD	0	0	0	0	7.74E−08	0	8.25E−05	2.21E−05	2.77E−08	4.39E−11	5.78E−05	447.2063
19	F19												
	Mean	0.002478	0.003114	8.59E−05	0.015298	0.010028	0.006603	0.018453	0.334147	0.000128	0.08692	0.003995	13.67791
	SD	0.000525	0.008358	2.95E−05	0.004101	0.004341	0.002143	0.018247	0.093514	7.84E−05	0.018087	0.001404	3.181055
20	F20												
	Mean	0	0.003114	0	0	0.000691	0	0.011967	102.5279	7.32E−07	9.36E−08	0.021563	79.47374
	SD	0	0.008358	0	0	0.000301	0	0.001819	280.5584	3.87E−06	3.65E−07	0.001391	6.227051
21	F21												
	Mean	0	0	0	0	1.4E−06	0	0.014547	9968.796	2.55E−06	3.4E−09	0.000358	284,631.1
	SD	0	0	0	0	1.35E−06	0	0.007048	13,638.56	8.92E−06	3.67E−09	6.46E−05	49,205.92
22	F22												
	Mean	27.34575	28.80641	0.414634	21.53581	19.09657	8.09052	56.7507	7.479017	4.987585	422.0499	40.02735	21,808,082
	SD	14.02782	22.78898	0.233254	0.223952	5.28918	2.543673	58.05154	14.97531	3.966018	381.7147	44.13148	6,757,573
23	F23												
	Mean	0.66667	0.66667	0.66667	0.66667	0.66667	0.6667	0.68292	0.667031	0.555667	0.841247	0.667355	156,144.5
	SD	0	0	0	0	0	0	0.039533	0.000305	0.252455	0.191106	0.000395	44,503.72
24	F24												
	Mean	0	51.63829	0	210.3432	50.8443	33.80478	31.14245	80.59337	1.54E−06	205.3158	39.93578	144.1689
	SD	0	16.08217	0	8.29743	11.83536	5.255376	10.1134	34.97928	1.75E−06	13.60851	10.75652	18.15777

Table 7 (continued)

No.	Functions	DE	PSO	SOS	ACO	FPA	CS	FA	BA	GA	ABC	IWO	BeeA
25	F25												
	Mean	0	0.010661	0	0	0.00396	0	0.001441	97.28893	0.024197	0.002563	21.63459	209.0089
	SD	0	0.01356	0	0	0.005355	0	0.001434	34.27794	0.023658	0.013976	11.60937	38.00338
26	F26												
	Mean	0	0.8089	0	0	0.032343	2.04E−06	0.005313	15.30773	0.000154	0.000868	13.05549	18.06537
	SD	0	0.873708	0	0	0.169818	7.67E−06	0.000395	1.281052	0.000316	0.001012	8.698354	0.343134
27	F27												
	Mean	− 1	− 1	− 0.90081	− 1	− 1	− 1	− 1	− 0.94186	− 0.98513	− 0.99997	− 1	− 1
	SD	0	0	0.052519	0	0	0	0	0.038068	0.027424	3.29E−05	0	0
28	F28												
	Mean	− 3.8628	− 3.8628	− 3.8518	− 3.8628	− 3.8628	− 3.8628	− 3.8628	− 3.8628	− 3.8628	− 3.8628	− 3.8628	− 3.8628
	SD	0	0	0.068996	0	0	0	0	0	0	0	0	0
29	F29												
	Mean	− 3.3224	− 3.8628	− 2.68162	− 3.3224	− 3.3224	− 3.3224	− 3.28267	− 3.27472	− 3.27869	− 3.3224	− 3.2032	− 3.3224
	SD	0	0	0.221309	0	0	0	0.057152	0.059394	0.058424	0	0	0
30	F30												
	Mean	− 10.1532	− 5.38903	− 1.59611	− 6.66531	− 10.1532	− 10.1532	− 9.81637	− 5.47161	− 6.48991	− 10.1532	− 6.7162	− 10.1532
	SD	0	3.098941	0.827992	3.792462	0	0	1.281835	3.26427	3.751506	0	3.190832	0
31	F31												
	Mean	− 10.4029	− 7.56765	− 1.65021	− 9.51602	− 10.4029	− 10.4029	− 10.4029	− 4.21061	− 8.39645	− 10.4029	− 8.16678	− 10.4029
	SD	0	3.577358	0.641299	2.091533	0	0	0	2.657822	3.38837	0	3.28606	0
32	F32												
	Mean	− 10.5364	− 7.88964	− 0.90081	− 10.5364	− 10.5364	− 10.5364	− 10.5364	− 5.46788	− 9.17502	− 10.5364	− 8.43238	− 10.5364
	SD	0	3.809105	0.052519	0	0	0	0	3.478352	2.7908	0	3.331965	0
33	F33												
	Mean	0.39789	0.39789	0.39789	0.39789	0.39789	0.3979	0.3979	0.39789	0.39789	0.39789	0.39789	0.39789
	SD	0	0	0	0	0	0	0	0	0	0	0	0
34	F34												
	Mean	3	3	3	3	3	3	3	3	3	3	3	3
	SD	0	0	0	0	0	0	0	0	0	0	0	0
35	F35												
	Mean	0.057734	0.099873	0.02659	0.09033	0.099873	0.0999	0.0999	3.326566	0.099873	0.099874	0.099873	0.106548
	SD	0.049063	0	0.037722	0.014239	0	0	0	1.091326	0	1.22E−06	0	0.044956
36	F36												
	Mean	0.099873	0.099873	0.087774	0.099873	0.099873	0.0999	0.0999	4.303233	0.199871	0.117795	0.099873	2.009894
	SD	0	0	0.028386	0	0	0	0	1.272923	0.064326	0.027295	0	0.887218
Average CPU time		38.05468	44.53103	5.934168	69.61049	14.94001	10.74201	73.82226	8.731651	28.87455	15.50262	14.44403	15.5324

Table 8 The best values obtained by the twelve algorithms for 100,000 function evaluation

Functions	DE	PSO	SOS	ACO	FPA	CS	FA	BA	GA	ABC	IWO	BeeA
F1	0	0	0	0	0	0	0	0	0	1.8E−09	0	1.92E−08
F2	− 1	− 1	− 1	− 1	− 1	− 1	− 1	− 1	− 1	− 1	− 1	− 1
F3	0	0	0	0	0	0	0	0	0	0	0	4.66E−08
F4	0	0	0	0	0	0	3.56E−09	0	0	0	0	2.39E−08
F5	0	0	0	0	0	0	0	0	0	0.000339	0	0
F6	− 1.8013	− 1.8013	− 1.8013	− 1.8013	− 1.8013	− 1.8013	− 1.8013	− 1.8013	− 1.8013	− 1.9992	− 1.8013	− 1.8013
F7	0	0	0	0	0	0	1.34E−05	0	0	1.32E−05	0.043671	2.41E−09
F8	− 1.0316	− 1.0316	− 1.0316	− 1.0316	− 1.0316	− 1.0316	− 1.0316	− 1.0316	− 1.0316	− 1.0316	− 1.0316	− 1.0316
F9	0	0	0	0	0	0	3.11E−09	0	0	0	0	3.28E−09
F10	0	0	0	0	0	0	2.34E−09	0	0	7.47E−09	0	1.21E−06
F11	− 186.731	− 186.731	− 186.731	− 186.722	− 186.731	− 186.731	− 186.731	− 186.731	− 186.731	− 186.718	− 186.731	− 186.731
F12	6.28E−06	0	6.79E−08	0.000115	2.57E−07	0	3.58E−07	3.72E−07	0.004927	0.004427	8.18E−07	0.000549
F13	− 4.6877	− 4.6877	− 4.6877	− 4.8502	− 4.6853	− 4.6877	− 4.6459	− 4.6459	− 4.6877	− 4.8539	− 4.6459	− 4.6877
F14	0	0	0	0	1.03E−07	5.04E−09	4.96E−07	8.7E−07	0	3.4141	2.23E−06	0.33372
F15	− 9.6602	− 9.5982	− 9.6602	− 6.5323	− 8.3868	− 9.1695	− 9.2908	− 7.2069	− 9.6602	− 7.1373	− 9.509	− 9.4139
F16	0	0	0	1.14E−05	0.03451	8.2E−06	2.02E−06	9.73E−06	0	0.012863	2.8E−05	47.6718
F17	0	0	0	0.001007	8.5222	0.0033	0.000789	5251.204	0	5.0431	4.42E−05	14,826.5
F18	0	0	0	0.000239	1.4368	0.000464	0.000371	0.000214	0	0.49538	0.000537	1741.767
F19	0.006722	0.00185	0.000135	0.044952	0.024414	0.0138	0.0021	0.67797	9.15E−05	0.1798	0.006527	5.2361
F20	0	3.69E−06	0	0.63899	3.9703	0.6389	0.0182	2.1845	0	3.4913	0.026579	64.2839
F21	0	0	0	0.010114	187.2261	0.0548	0.0237	41,044.63	0	57.6019	1080.18	206,298.8
F22	23.95	0.013291	16.0368	555.04	388.9691	28.6319	26.6913	0	0.17648	109,277.4	25.3561	5,945,167
F23	0.66667	0.66667	0.66667	3.6897	5.2938	0.6921	0.6669	0.66687	0.004764	715.185	0.66728	63,678.79
F24	25.6669	27.8588	0	193.0158	60.5309	62.8184	13.9298	30.8466	0	182.0688	27.8689	128.4608
F25	0	0	0	0.010013	1.1477	0.0176	0.0016	75.8348	0	1.0316	271.126	131.0838
F26	0	0	0	0.010273	2.1005	1.8561	0.0072	12.6763	4.13E−09	2.5688	0.005582	17.4312
F27	− 1	− 1	− 0.93624	− 1	− 1	− 1	− 1	− 1	− 1	− 1	− 1	− 1
F28	− 3.8628	− 3.8628	− 3.8551	− 3.8628	− 3.8628	− 3.8628	− 3.8628	− 3.8628	− 3.8628	− 3.8628	− 3.8628	− 3.8628
F29	− 3.3224	− 3.3224	− 3.1386	− 3.3224	− 3.3224	− 3.3224	− 3.3224	− 3.2032	− 3.3224	− 3.3224	− 3.2032	− 3.3224
F30	− 10.1532	− 10.1532	− 7.6019	− 10.1532	− 10.1532	− 10.1532	− 10.1532	− 10.1532	− 10.1532	− 10.1532	− 10.1532	− 10.1532
F31	− 10.4029	− 10.4029	− 3.9141	− 10.4029	− 10.4029	− 10.4029	− 10.4029	− 10.4029	− 10.4029	− 10.4029	− 10.4029	− 10.4029
F32	− 10.5364	− 10.5364	− 5.3774	− 10.5364	− 10.5364	− 10.5364	− 10.5364	− 10.5364	− 10.5364	− 10.5364	− 10.5364	− 10.5364
F33	0.39789	0.39789	0.39789	0.39789	0.39789	0.3979	0.3979	0.39789	0.39789	0.39789	0.39789	0.39789
F34	3	3	3	3	3	3	3	3	3	3	3	3
F35	0.099873	0.099873	0.099873	0.099873	0.022783	0.0999	0.0999	1.5999	0.099873	0.099873	0.099873	0.00315
F36	0.099873	0.099873	0.099873	0.099873	0.19987	0.1999	0.0999	3.2999	0.099873	0.14938	5.3999	0.29987

Table 9 The best values obtained by the twelve algorithms for 300,000 function evaluation

Functions	DE	PSO	SOS	ACO	FPA	CS	FA	BA	GA	ABC	IWO	BeeA
F1	0	0	0	0	0	0	0	0	0	0	0	0
F2	− 1	− 1	− 1	− 1	− 1	− 1	− 1	− 1	− 1	− 1	− 1	− 1
F3	0	0	0	0	0	0	0	0	0	0	0	0
F4	0	0	0	0	0	0	1.05E−08	0	0	0	0	0
F5	0	0	0	0	0	0	0	0	0	4.77E−07	0	0
F6	− 1.8013	− 1.8013	− 1.8013	− 1.8013	− 1.8013	− 1.8013	− 1.8013	− 1.999	− 1.8013	− 1.9995	− 1.8013	− 1.8013
F7	0	0	0	0	0	0	4.65E−05	0	0	3.81E−05	0.043671	0
F8	− 1.0316	− 1.0316	− 1.0316	− 1.0316	− 1.0316	− 1.0316	− 1.0316	0	− 1.0316	− 1.0316	− 1.0316	− 1.0316
F9	0	0	0	0	0	0	0	0	0	0	0	0
F10	0	0	0	0	0	0	1.9E−09	0	0	3.42E−09	0	0
F11	− 186.731	− 186.731	− 186.73	− 186.731	− 186.731	0	− 186.731	− 186.731	− 186.731	− 186.723	− 186.731	− 186.731
F12	0	0	0	0	0	0	3.6E−07	2.2E−08	6.85E−06	0.001101	3.77E−07	0.000156
F13	− 4.6877	− 4.6877	− 4.6877	− 4.8438	− 4.6877	− 4.6877	− 4.6877	− 4.6877	− 4.6877	− 4.7598	− 4.6877	− 4.6877
F14	0	0	0	0	0	0	3.58E−07	5.37E−07	0.002573	1.2659	1.14E−06	0.14506
F15	− 9.6602	− 9.6176	− 9.6602	− 7.6773	− 9.2069	− 9.5651	− 9.4998	− 8.306	− 9.6602	− 6.7529	− 9.618	− 8.9875
F16	0	0	0	0	7.61E−07	0	1.28E−06	8.65E−06	0	3.51E−08	1.79E−05	51.1072
F17	0	0	0	0	0.000251	0	0.000378	230.4592	0	6.12E−06	1.91E−05	16,292.04
F18	0	0	0	0	1.05E−05	0	9.23E−05	0.000131	0	1.13E−06	0.000325	1558.044
F19	0.001077	0.001036	6.19E−05	0.012237	0.007703	0.006	0.00096	0.27953	6.06E−05	0.049673	0.002275	7.4054
F20	0	1.6E−05	0	0	0.030609	9.15E−07	0.0096	0.14594	0	9.11E−06	0.01775	66.0809
F21	0	0	0	0	0.003907	0	0.0083	8668.752	0	0.0001	0.000286	200,222.4
F22	19.4018	4.569	1.2845	24.3582	15.4053	7.4802	25.4522	9.9913	0.18799	838.8028	22.8844	1,2049,140
F23	0.66667	0.66667	0.66667	0.66667	0.66672	0.6667	0.6669	0.6668	0.002161	3.3742	0.66692	41,617.61
F24	0	24.7571	0	176.2535	34.7669	34.993	21.8894	32.8358	0	190.1302	25.8747	90.541
F25	0	0	0	0	0.003783	0	0.000951	44.6566	0	0.00255	43.5766	157.1165
F26	0	0	0	0	0.0057	1.47E−06	0.0046	13.2171	0	0.017869	0.003726	17.3179
F27	− 1	− 1	− 0.93623	− 1	− 1	− 1	− 1	− 1	− 1	− 1	− 1	− 1
F28	− 3.8628	− 3.8628	− 3.8523	− 3.8628	− 3.8628	− 3.8628	− 3.8628	− 3.8628	− 3.8628	− 3.8628	− 3.8628	− 3.8628
F29	− 3.3224	− 3.3224	− 3.0159	− 3.3224	− 3.3224	− 3.3224	− 3.3224	− 3.3224	− 3.3224	− 3.3224	− 3.2032	− 3.3224
F30	− 10.1532	− 10.1532	− 3.5338	− 10.1532	− 10.1532	− 10.1532	− 10.1532	− 10.1532	− 10.1532	− 10.1532	− 10.1532	− 10.1532
F31	− 10.4029	− 10.4029	− 6.3845	− 10.4029	− 10.4029	− 10.4029	− 10.4029	− 10.4029	− 10.4029	− 10.4029	− 10.4029	− 10.4029
F32	− 10.5364	− 10.5364	− 4.9763	− 10.5364	− 10.5364	− 10.5364	− 10.5364	− 10.5364	− 10.5364	− 10.5364	− 10.5364	− 10.5364
F33	0.39789	0.39789	0.39789	0.39789	0.39789	0.3979	0.3979	0.39789	0.39789	0.39789	0.39789	− 10.5364
F34	3	3	3	3	3	3	3	3	3	3	3	3
F35	0.000614	0.099873	0	0.026069	0.099873	0.0999	0.000209	1.2999	0.099873	0.099873	0.39789	0.099873
F36	0.099873	0.099873	0.099873	0.099873	0.099873	0.0999	0.0999	3.0999	0.099873	0.099922	0.099873	0.29987

Table 10 The best values obtained by the twelve algorithms for 500,000 function evaluation

Functions	DE	PSO	SOS	ACO	FPA	CS	FA	BA	GA	ABC	IWO	BeeA
F1	0	0	0	0	0	0	0	0	0	0	0	0
F2	− 1	− 1	− 1	− 1	− 1	− 1	− 1	− 1	− 1	− 1	− 1	− 1
F3	0	0	0	0	0	0	0	0	0	0	0	0
F4	0	0	0	0	0	0	1.84E−09	0.88281	0	0	0	0
F5	0	0	0	0	0	0	0	0	0	1.85E−05	0	0
F6	− 1.8013	− 1.8013	− 1.8013	− 1.8013	− 1.8013	− 1.8013	− 1.8013	− 1.9948	− 1.8013	− 1.9999	− 1.8013	− 1.8013
F7	0	0	0	0	0	0	4.17E−05	0	0	1.94E−05	0.043671	0
F8	− 1.0316	− 1.0316	− 1.0316	− 1.0316	− 1.0316	− 1.0316	− 1.0316	− 1.0316	− 1.0316	− 1.0316	− 1.0316	− 1.0316
F9	0	0	0	0	0	0	0	0.59926	0	0	0	0
F10	0	0	0	0	0	0	5.45E−09	0	0	3.26E−09	0	0
F11	− 186.731	− 186.731	− 186.731	− 186.73	− 186.731	− 186.731	− 186.731	− 186.731	− 186.731	− 186.73	− 186.731	− 186.731
F12	0	0	0	0	0	0	9.72E−08	6.83E−08	0.000496	0.000635	3.79E−07	0.000142
F13	− 4.6877	− 4.6877	− 4.6877	− 4.8706	− 4.6877	− 4.6877	− 4.6877	− 4.5249	− 4.6877	− 4.7079	− 4.6877	− 4.6877
F14	0	0	0	0	0	0	2.01E−07	3.89E−07	0.000374	0.80106	1.39E−06	0.10804
F15	− 9.6602	− 9.6552	− 9.6602	− 7.956	− 9.4107	− 9.6525	− 9.5827	− 7.3601	− 9.6602	− 7.3602	− 9.5422	− 9.1805
F16	0	0	0	0	0	0	8.79E−07	7.94E−06	0	0	1.87E−05	46.3117
F17	0	0	0	0	9.42E−09	0	0.000326	1.33E−05	0	0	1.67E−05	16,357.61
F18	0	0	0	0	0	0	4.77E−05	0.000119	0	0	0.00027	1617.809
F19	0.001082	1.37E−05	3.83E−05	0.010425	0.003261	0.0028	0.0017	0.13126	3.45E−05	0.051952	0.001808	9.0299
F20	0	1.37E−05	0	0	0.000176	0	0.0093	0.20132	0	0	0.016854	64.1924
F21	0	0	0	0	5.56E−08	0	0.0072	41.0452	0	0	0.000178	185,834.8
F22	19.661	0.000934	0.015588	21.1202	0.41966	0.3633	25.3938	0.041286	0.006881	61.3095	21.1829	5,954,469
F23	0.66667	0.66667	0.66667	0.66667	0.66667	0.6667	0.6668	0.66679	0.000509	0.66788	0.66692	84,415.13
F24	0	22.884	0	193.0233	24.6517	22.2855	16.9146	31.8405	0	163.6747	23.8832	100.4905
F25	0	0	0	0	7.06E−08	0	0.000589	23.3498	0	2.6E−07	6.0416	128.3801
F26	0	0	0	0	3.22E−05	0	0.0046	12.2601	0	5.05E−05	0.003259	17.2201
F27	− 1	− 1	− 0.99716	− 1	− 1	− 1	− 1	− 1	− 1	− 1	− 1	− 1
F28	− 3.8628	− 3.8628	− 3.8522	− 3.8628	− 3.8628	− 3.8628	− 3.8628	− 3.8628	− 3.8628	− 3.8628	− 3.8628	− 3.8628
F29	− 3.3224	− 3.3224	− 2.9675	− 3.3224	− 3.3224	− 3.3224	− 3.3224	− 3.3224	− 3.3224	− 3.3224	− 3.2032	− 3.3224
F30	− 10.1532	− 10.1532	− 3.1178	− 10.1532	− 10.1532	− 10.1532	− 10.1532	− 10.1532	− 10.1532	− 10.1532	− 10.1532	− 10.1532
F31	− 10.4029	− 10.4029	− 3.4116	− 10.4029	− 10.4029	− 10.4029	− 10.4029	− 10.4029	− 10.4029	− 10.4029	− 10.4029	− 10.4029
F32	− 10.5364	− 10.5364	− 0.99716	− 10.5364	− 10.5364	− 10.5364	− 10.5364	− 10.5364	− 10.5364	− 10.5364	− 10.5364	− 10.5364
F33	0.39789	0.39789	0.39789	0.39789	0.39789	0.3979	0.3979	0.39789	0.39789	0.39789	0.39789	0.39789
F34	3	3	3	3	3	3	3	3	3	3	3	3
F35	0.000162	0.099873	0	0.045263	0.099873	0.0999	0.0999	0.89987	0.099873	0.099873	0.099873	1.5E−10
F36	0.099873	0.099873	0	0.099873	0.099873	0.0999	0.0999	2.2999	0.099873	0.099877	0.099873	0.49987

Table 11 Number of functions where each algorithm reaches the global minimum

FE	DE	PSO	SOS	ACO	FPA	CS	FA	BA	GA	ABC	IWO	BeeA
100,000	29	29	26	21	20	22	17	22	29	15	20	15
300,000	30	29	28	29	23	30	18	19	29	16	20	20
500,000	30	29	26	29	24	30	16	17	29	21	19	20

Table 12 Summary of algorithms percentage success ratio (%)

FE	DE (%)	PSO (%)	SOS (%)	ACO (%)	FPA (%)	CS (%)	FA (%)	BA (%)	GA (%)	ABC (%)	IWO (%)	BeeA (%)
100,000	11	11	10	8	8	8	6	8	11	6	8	6
300,000	10	10	10	10	8	10	6	7	10	5	7	7
500,000	10	10	9	10	8	10	6	6	10	7	7	7

Table 13 Friedman's test results for the twelve algorithms. Here "Average" indicates Friedman mean rank, "Statistic" indicates Friedman statistic, and "p values" indicates the corresponding p values

SN	Algorithm	Average (100,000 FE)	Algorithm	Average (300,000 FE)	Algorithm	Average (500,000 FE)
1	PSO	4.58	DE	4.97	DE	5.07
2	GA	4.64	GA	5.10	PSO	5.19
3	DE	4.79	PSO	5.24	GA	5.29
4	SOS	5.47	ACO	5.49	ACO	5.76
5	ACO	6.61	SOS	5.85	SOS	5.79
6	CS	6.67	CS	6.14	CS	5.86
7	FA	6.85	FPA	6.65	FPA	6.14
8	FPA	7.07	FA	7.36	ABC	7.28
9	IWO	7.07	IWO	7.39	IWO	7.57
10	BA	7.31	BA	7.72	FA	7.82
11	ABC	8.06	ABC	7.86	BA	8.04
12	BeeA	8.89	BeeA	8.24	BeeA	8.18
Statistics		76.827		93.327		79.488
p values		0.0001		0.0001		0.0001

mentioned above). As presented in Table 13, analysis of the statistical result shows that the tabulated mean rank results (using the average global minimum values) show a statistically significant difference among the compared algorithms. The obtained Friedman's statistics for the three number of function evaluations are 76.827, 93.327 and 79.488 with corresponding p values of 0.0001 each, which is less than 0.05. Therefore, the results strongly indicate that there are statistically significant differences in the algorithms' performances. In summary, DE is the best with average ranks of 4.79, 4.97 and 5.07, PSO is the second best with average ranks of 4.58, 5.19 and 5.24 and GA being the third best with average ranks of 4.64, 5.10 and 5.29, respectively. These results are somewhat consistent with those shown in Tables 5, 6, 7, 8, 9, 10, 11 and 12.

Additional statistical tests were conducted because the result of the Friedman's test only showed an indication that a statistically significant difference exist among the compared algorithms, but the test fails to specifically point out where the actual differences exist. Therefore, to determine exactly where the difference exist among the twelve compared algorithms, a post hoc analysis with Wilcoxon signed-rank tests with a Bonferroni correction was applied, resulting in a significance level set at $p < 0.0042$. The results of the post hoc tests clearly showing the statistically significant differences among the algorithms are shown in Table 14. Furthermore, the results specifically show all the similarity mapping in terms of which set of algorithms are statistically significantly different in performance using the average global minimum values obtained by running the 36 test problems. The multiple comparisons carried out for the

post hoc tests involves using first DE as the control algorithm to evaluate the performance differences of other algorithms and subsequently repeating the same tabulation for the remaining algorithms using different control algorithms. Summarising the entire statistical evaluation, a conclusion can therefore be drawn that while there were not any noticeable statistically significant differences among the performances of DE, PSO, GA and SOS, some significant performance differences were observed among the set of algorithms presented in Table 14. However, as shown in this paper the robust statistical tests conducted indicate that the performances of the DE, PSO, and GA methods appear to be the best performed algorithms, and they are closely followed by SOS, CS, and ACO algorithms. However, a generalised conclusion cannot be drawn at this level because a slight change in parameter tuning values can drastically result in significant performance changes for any of the algorithms, excluding those methods that do not have any control parameters such as the SOS. It is noteworthy to mention that the scope of the current study does not cover real-parameter optimisation for obtaining best algorithm performance, but rather its primary focus is to enumerate the possible performance differences that exist among the different standard metaheuristic algorithms.

5 Conclusions and future study

This paper presented detailed discussions on the algorithmic concepts of some selected well-known metaheuristic algorithms with specific emphases on motivations, design concepts, advantages and disadvantages of each of the algorithms' implementation techniques. Furthermore, a comparative study of the representative global optimisation algorithms was similarly presented using both descriptive and in-depth statistical analysis techniques. The statistical analysis results revealed that the success ratios of the DE, PSO, GA, SOS, CS, and ACO are better than FPA, FA, ABC, BA, IWO and BeeA. However, DE, PSO and GA exhibited excellent performances across all the test problems, each achieving a maximum of 11% success ratio. In addition, SOS and CS also achieved good performances with 10% success ratios. Furthermore, the effect of the maximum number of function evaluation on the performances of each algorithm was demonstrated. The varying number of function evaluations used as the algorithm stopping criteria were 100,000 FE, 300,000 FE and 500,000 FE. It was observed that some sets of algorithms performed better with a larger number of FEs such as DE, ACO and CS, while some produced good results with a small number of FEs such as BA and IWO. However, in some instances, the number of FEs did not affect the performance of the algorithm, as it was the case with the PSO and GA algorithms.

In the domain of global optimisation and metaheuristic enthusiasts, researchers disposed to the application of various evolutionary algorithms in solving real-world problems would always want to apply the best of the existing or new algorithms to solve their optimisation problems. However, the judgement of selecting the best out of the several existing metaheuristic algorithms is usual anticipated from the concluding remarks drawn by the inventors of the algorithms. In most cases, thorough evaluation and analyses are not conducted prior to users making selection as to what choice of algorithm would be better for their type of problem. Interestingly, the results obtained from this study have shown that, in most cases, the individual algorithm performances are greatly affected by a number of factors, among which includes: algorithm control parameters, number of function evaluations, design and implementation structures, to mention but a few. From the output of this study also, it is noted that some of the algorithms performed equally under the same experimental conditions and test problems. Finally, a suggestion for future research directions is the development of a statistical criterion for optimum parameter tuning and selection to obtain the best performance for each algorithm.

Compliance with ethical standards

Conflict of interest The authors declare that there is no conflict of interests regarding the publication of this paper.

Appendix 1: Algorithmic concepts

Here, a summarised description of the algorithmic concepts of each of the twelve selected metaheuristics is presented. The discussion covers background information, design concepts, motivations, and the advantages and disadvantages of each algorithm implementation approach. However, for further study, interested readers are referred to literature related to the primary source information of each algorithm. For details, see [6, 9–19, 24, 25].

Cuckoo search algorithm

Cuckoo search algorithm, is a SI-based algorithm, inspired by the parasitic behaviour of some cuckoo birds species together with the Levy flight behaviour of some birds species and fruit flies [37, 38]. Some cuckoo bird species, specifically, the brood parasite species, depend on other birds to hatch their eggs and care for their young. These

Table 14 The significantly different algorithms according to the output of the Friedman Test (with post hoc tests) for twelve algorithms using average global minimum values

FE	DE	PSO	SOS	ACO	FPA	CS	FA	BA	GA	ABC	IWO	BeeA
100,000	ACO	ACO	ACO	DE	DE	DE	DE	DE	ACO	DE	DE	DE
	CS	CS	FPA	PSO	PSO	PSO	PSO	PSO	CS	PSO	PSO	PSO
	FPA	FPA	CS	GA	SOS	SOS	SOS	SOS	FPA	SOS	SOS	SOS
	FA	FA	FA	FA	GA	GA	CS	GA	FA	GA	GA	GA
	BA	BA	BA	ABC	FA	ABC	GA	FA	BA	ACO	ABC	ACO
	ABC	ABC	ABC	BeeA	ABC	BeeA	ACO	ABC	ABC	FPA	BeeA	FPA
	IWO	IWO	IWO		BeeA	FA	BA	BeeA	IWO	CS	FA	CS
	BeeA	BeeA	BeeA				IWO		BeeA	BA		BA
										IWO		IWO
300,000	FPA	FPA	FA	FPA	DE	FPA	DE	DE	FPA	DE	DE	DE
	FA	FA	BA	FA	PSO	FA	PSO	PSO	FA	GA	PSO	PSO
	BA	BA	ABC	BA	SOS	BA	SOS	SOS	BA	PSO	SOS	SOS
	ABC	ABC	IWO	ABC	ACO	ABC	ACO	GA	ABC	SOS	GA	GA
	IWO	IWO	BeeA	IWO	CS	IWO	FPA	ACO	IWO	CS	ACO	CS
	BeeA	BeeA		BeeA	FA	BeeA	CS	CS	BeeA	ACO	CS	ACO
					GA		GA					
					ABC							
500,000	FPA	FPA	FA	FPA	DE	FPA	DE	DE	FPA	DE	DE	DE
	FA	FA	BA	FA	PSO	FA	PSO	PSO	FA	GA	PSO	PSO
	BA	BA	ABC	BA	ACO	BA	SOS	SOS	BA	PSO	SOS	SOS
	ABC	ABC	IWO	ABC	CS	ABC	CS	CS	ABC	SOS	GA	GA
	IWO	IWO	BeeA	IWO	FA	IWO	GA	GA	IWO	CS	ACO	CS
	BeeA	BeeA		BeeA	BA	BeeA	ACO	ACO	BeeA	ACO	CS	ACO
					ABC		FPA	FPA				
					GA							
					IWO							
					BeeA							

species lay their eggs in the nest of birds that contain newly hatched eggs. Usually, the eggs of cuckoo birds hatch faster than the eggs of their host birds, and a cuckoo chick grows at a faster pace. Hence, by nature, newly hatched cuckoo chicks remove all the eggs or chicks from the nest of the host bird, with the aim of maximising the quantity of food that it can get from the host bird. This parasitic behaviour of cuckoo birds inspired Yang and Deb [13] to propose the CS algorithm. The following rules were considered in the design of the CS algorithm:

1. Every cuckoo bird lays a single egg per time, and randomly distributes its eggs among different hosts.
2. Nests with high-quality eggs will stay alive for the next generation.
3. There is a fixed number of nests. Also, cuckoo eggs are seen by their host bird by a likelihood of $p_a \in [0, 1]$. Immediately a strange egg is discovered, a host bird can either abandon its nest or remove the egg from the nest. The position of each cuckoo is generated by performing a Levy flight as shown in Eq. (1).

$$X_i^{(t-1)} = X_i^{(t)} + \alpha \oplus \text{Levy}(\lambda) \tag{1}$$

The step size $\alpha > 0$ is determined by the scales of the problem at hand. The symbol $\oplus$ denotes entry wise multiplication. Levy flight provides random walks, and it has a Levy distribution shown in Eq. (2). The Levy distribution has an unlimited variance and unlimited mean.

$$\text{Levy} \sim u = t^{-\lambda} \quad (1 < \lambda \leq 3) \tag{2}$$

The classic CS algorithm was designed to handle problems in a continuous space, however, as shown in Eq. (3), the algorithm can be used to handle continuous problems. To achieve this, each cuckoo position can be converted to a binary value of 0 or 1 using the sigmoid function.

$$S(V_i^t) = \frac{1}{1 + e^{-V_i^t}}, \tag{3}$$

Therefore, in place of Eq. (3), each cuckoo position is updated by using Eq. (4):

$$X_i^t = \begin{cases} 1 & \text{if rand}() \leq S(V_i^t), \\ 0 & \text{otherwise}, \end{cases} \tag{4}$$

rand() is a random number drawn uniformly from the interval [0, 1]. The following pseudocode outlined in Algorithm 1 reflects the above explanation.

Algorithm 1: Cuckoo Algorithm via Levy Flight [13]

Objective function $f(x), x = (x_1, \ldots, x_D)^T$
Generate an initial population of n host nests x_i $(i = 1, 2, \ldots, D)$
While $(t < MaxGeneration) or (stop\ crieterion)$
    Get a cuckoo randomly Levy flight
    Evaluate its quality or fitness f_i
    Choose a nest among n (say, j) randomly
    If $(F_i > F_j)$
        Replace j by the new solution;
    End if
    A fraction of worse nests is abandoned and new ones are built;
    Keep the best solutions (or nest with quality solutions);
    Rank the solutions and find the current best; and,
    Rank the best and find the current best x_*
End while
Post process result and visualisation

Firefly algorithm

The firefly algorithm is inspired by the flashing light attribute of fireflies. There are different species of fireflies and the majority of these species produce short flashlights at regular intervals or time breaks. These flashlights act as signals to entice other fireflies and also to send warnings to potential prey. FA is mostly suited for tackling difficult NP-hard problems [6, 28]. As shown in Eq. (5), the light intensity of a flashlight is inversely proportional to the square of its distance. That is, light intensity reduces per increase in distance, and this is because, as distance increases, flashlight is released into the atmosphere.

$$I \propto {}^{1}/_{r^2} \tag{5}$$

Flashlight can be formulated in a manner that it will be proportional to the fitness function value to be optimised. The following design rules were used to design FA [39]:

1. All firefly species are identical in sex.
2. The attractiveness of a firefly is proportional to the light intensity produced by the firefly. This indicates that firefly that produces a low flashlight will be attracted to fireflies that produces high flashlight.
3. The intensity of flashlight produced by a firefly is determined by the fitness function landscape that is to be improved.

Two important concerns that must be properly considered when using FA are light intensity and attractiveness. Usually, for maximum optimisation problems, light intensity (I), produced at a specified point (y), is directly proportional to the fitness value of the fitness function (that is, $I(y) \propto F(y)$). As shown in Eq. (6), light intensity changes with respect to distance and intensity of light emitted into the atmosphere.

$$I(r) = I_0 e^{-\gamma r^2} \tag{6}$$

where I_0 refers to the initial light intensity at $r = 0$, γ represents the light absorption coefficient, and r denotes the distance. As shown in Eq. (6), by combining the effect of the inverse square law and absorption, the singularity at $r = 0$ is circumvented in the expression $1/r^2$ [39]. In addition, as shown in Eq. (7), the singularity is also evaded by approximating them in Gaussian form. Furthermore, as shown in Eq. (7), the attractiveness of a firefly (β) is proportional to the light intensity of the firefly.

$$\beta = \beta_0 e^{-\gamma r^2} \tag{7}$$

where β_0 refers to the attractiveness at $r = 0$.

The Euclidian distance between two fireflies (x_i and x_j) is given by:

$$r_{ij} = \| x_i - x_j \| = \sqrt{\sum_{k=1}^{d} \left(x_{i,k} - x_{j,k}\right)^2} \tag{8}$$

where d is the dimension of the problem. Movement of a firefly from point i to point j is shown in Eq. (9):

$$x_i = x_i + \beta_0 e^{-\gamma r_{ij}^2} \left(x_j - x_i\right) + \alpha \epsilon_i \tag{9}$$

where $\alpha \in [0, 1], \gamma \in [0, \infty)$. ϵ_i is a random number obtained from a Gaussian distribution. ϵ_i can be replaced with $\text{rand} - 0.5$, where $\text{rand} \in [0, 1]$. The third term ($\alpha \epsilon_i$) in Eq. (9) shows a firefly movement from one point to another, with regards to their attractiveness. The following outline presented in Algorithm 2 reflects the above explanation.

Algorithm 2: Firefly Algorithm [39]

Define initial values of firefly parameters: NF, NG, β_o, α, and γ
Define Fitness function $f(x), x = (x_1, x_2 \ldots x_D) t$
Initialise n positions of firefly $(i = 1, 2, 3, \ldots, n)$
Evaluate $f(x)$ to determine light intensity L_i of firely x_i
while (j < NG)
  for k = 1 to NF
    for m = 1 to NF
      If ($L_k < L_m$)
        Move firefly k towards firefly m
      end if
      Calculate attractiveness variance with distance r using exp(-γr)
      Calculate new fitness values for all fireflies
      Update firefly light intensity L_i
    end for
  end for
end while
Output optimised firefly light intensity

Ant colony optimisation

The ACO is a well-known metaheuristic, inspired by the foraging behaviour of some specific species of ant. The earliest form of ACO known as the Ant System seeks to search for an optimal path within a graph. Generally, to minimise time, ants aim to identify the best path between their nest and different food sources. Ants achieve this by communicating with each other through a chemical substance called pheromones. The movement of an ant from node x to y is measured by a probabilistic rule as shown in Eq. (10) [11, 40].

$$p_{i,j}^{(n)} = \frac{\left(\tau_{x,y}^{n-1}\right)^{\alpha}\left(\eta_{x,y}\right)^{\beta}}{\sum_{j\epsilon\Omega_x}\left(\tau_{x,y}^{n-1}\right)^{\alpha}\left(\eta_{x,y}\right)^{\beta}}, \quad \text{if } y \in \Omega_x \tag{10}$$

where $\tau_{a,b}^{n-1}$ refers to the pheromone deposited on an arc linking node x to node y, Ω_x refers to the collection of all the possible nodes a single ant can visit, provided the ant is located at node x; α regulates the effect of information provided by the pheromone, β regulates the heuristic information. α and β are constant values. $\eta_{x,y}$ refers to the heuristic information for an ant moving from node x to y, and it is a constant value for every iteration. The pheromone matrix is usually updated in two stages. As shown in Eq. (11), the initial update is performed after each ant has been moved for every construction step. The second update is performed after the entire ants in the colony have been moved using Eq. (12) [41].

$$\tau_{x,y}^{n-1} = \begin{cases} (1-\rho) * \tau_{x,y}^{n-1} + \rho * \Delta_{x,y}^{(k)}, & \text{if } (x,y) \text{is currently visited} \\ \tau_{x,y}^{n-1}, & \text{otherwise} \end{cases} \tag{11}$$

where ρ is the evaporation rate.

$$\tau^{(n)} = (1-\psi) * \tau^{(n-1)} + \psi * \tau^{(0)}, \tag{12}$$

where ψ is the pheromone decay coefficient (Fig. 5).

The main procedure for implementing the ACO, which is described above is presented in Algorithm 3.

Algorithm 3: Ant Colony Optimisation [42]

While (termination condition is not reached)
Initialise solution
Update pheromones
Perform daemon actions
End while

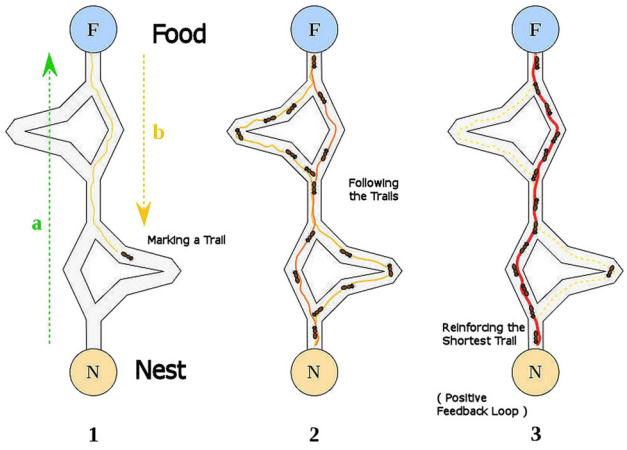


Fig. 5 Description of ant colony optimisation [73]

Bat algorithm

The BA is a metaheuristic based on the echolocation behaviour of bats. Most bat species utilise echolocation to search for food or prey. They also use echolocation to protect themselves from obstacles and to detect their perch in the midst of darkness. Bats produce high-volume sounds in patterns, and they are very attentive to echoes that may be redirected back from various entities in their environment [15]. When a bat is hunting for a prey, it produces different pulses at a very high frequency, and the frequency decreases as the bat moves closer to the prey [43]. Some bat species have the ability to see clearly, while some have the ability to smell properly. These senses in turn enhance their ability to properly identify a potential prey and to avoid possible impediments. The following rules were used to formulate BA:

1. Echolocation is used by all bats to identify distance. Furthermore, all bats can distinguish between obstacles and different kinds of prey.
2. In search for prey, Bats fly randomly, from position x_i and with velocity v_i at a constant frequency $f_{\min}$, changing wavelength λ and loudness A_o. Depending on the closeness of a potential prey, bats can adjust their pulse emission frequency, and wavelength.
3. The value for Loudness changes from a large positive number, A_o, to a small number, $A_{\min}$.
4. For every bat, the following are initialised: the position x_i, the velocity v_i and the frequency f_i. Afterwards, as shown in Eqs. (13)–(15), x_i, v_i and f_i are modified as follows [44]:

$$f_i = f_{\min} + (f_{\max} - f_{\min})\beta, \tag{13}$$

$$V_i^t = V_i^{t-1} + (X_i^t - X_*)f_i, \tag{14}$$

$$X_i^t = X_i^{t-1} + V_i^t \tag{15}$$

where β is a random number between 0 and 1, and X_* refers to the up-to-date global best solution. Furthermore, f_i is used to regulate the range and speed of bat movements. At the first iteration of the algorithm, all bats are assigned random frequency values, from $[f_{\min}, f_{\max}]$. According to Eq. (16), when the best solution is selected from the current set of generated solutions, the algorithm uses random walks to generate a new solution for each bat in the population:

$$X_{\text{new}} = X_{\text{old}} + \in A^t \tag{16}$$

where $\in$ is a randomly generated number between $[-1, 1]$, and A^t is the bat loudness at every time interval. Furthermore, at each iteration, the loudness and pulse rate emission are regulated as shown in Eqs. (17) and (18):

$$A_i^{t+1} = \propto A_i^t, \tag{17}$$

$$r_i^{t+1} = r_i^0[1 - \exp(-\gamma t)] \tag{18}$$

where $\propto$ and γ are BA parameters. Algorithm 4 describes the above explanation stepwise.

Algorithm 4: Bat Algorithm [15]

Objective function $f(x), x = (x_1, \ldots, x_D)^T$
Initialise Bat population $x_i (i = 1, 2, \ldots, n)$ *and* v_i
Define pulse frequency f_i *at* x_i
Initialise pulse rates r_i and the loudness A_i
While $t <$ *Max number of iterations*
  Generate new solutions by adjusting frequency and updating velocities and solutions
  If $(rand < r_i)$
    Select a solution among the best solutions
    Generate a local solution around the selected best solution
  End if
  Generate a new solution by flying randomly
  If $(rand < A_i$ *and* $f(x_i) < f(x_*))$
    Accept the new solution
    Increase r_i *and reduce* A_i
  End if
  Rank the bats and find the current best x_*
End while
Post process result and visualisation

Flower pollination algorithm

The FPA is a recent bio-inspired algorithm that mimics the pollination process of flowering plants. Pollination can be defined as the transmission of pollen grain from the (male) anther of a flowering plant to the (female) stigma of a flowering plant. The primary aim of flower pollination is to increase the number plants and the number of fit plants. There are different kinds of flower pollinators, some of which are mosquitoes, bee flies, butterflies, bats, water, birds and wind. Some pollinators exclusively pollinate specific flower species, thus avoiding other flower species; this is called flower constancy [45, 46]. Generally, there are two types of flower pollination, namely, biotic and abiotic pollination. In biotic pollination, pollinators are usually required to move the pollen grains from one flower to another, while in abiotic pollination, pollinators are not required. Water and wind can stand as abiotic pollinators. Some pollinators (called global pollinators) can fly over long distances and consequently cause pollination to occur between flowers that are in far proximity from each other. Global pollination generally ensures the reproduction of fit flowers [46]. Two types of pollination exist, namely, cross-pollination and self-pollination. Cross-pollination can be defined as the movement of pollen grains from the anther of a flowering plant to the stigma of another plant. On the other hand, self-pollination is the movement of pollen grains from the anther of a flower to the stigma of the same flower. Based on the above-mentioned pollination attributes, FPA was formulated by Yang [17] using four rules as outlined below:

1. The processes involved in biotic pollination and cross-pollination is regarded as global pollination, where global pollinators perform Levy flight.
2. Abiotic pollination and self-pollination are regarded as local pollination.
3. Flower constancy is also known as reproduction probability. Flower constancy is proportional to the resemblance between two different flowers that are to be pollinated.
4. FPA uses a switch probability $p \in [0, 1]$ to regulate global and local pollinations. Due to the closeness of some pollinators, a substantial fraction of the switch probability can be assigned during local pollination.

Rule 1 and flower constancy can be symbolised, as shown in Eq. (19).

$$x_i^{t+1} = x_i^t + L\left(x_i^t - g_*\right), \tag{19}$$

where X_i^t represents a vector x_i at different iterations t, and g_* represents the current best solution in iteration t. Furthermore, L represents Levy flight, and it can be obtained from a Levy distribution as shown in Eq. (20).

$$L \sim \frac{\lambda \Gamma(\lambda) \sin\left(\pi\lambda/2\right)}{\pi} \frac{1}{S^{1+\lambda}}, (s \gg s_0 > 0), \tag{20}$$

where $\Gamma(\lambda)$ represent a standard gamma function, which is only valid for large steps, $s > 0$.

Moreover, Rule 2 and flower constancy can be symbolised by Eq. (21).

$$x_i^{t+1} = x_i^t + \in \left(x_j^t - x_k^t\right), \tag{21}$$

where x_j^t represent pollen j at iteration t, x_k^t represent pollen k at iteration t. x_j^t and x_k^t represent pollen grains from different flowers. $\in$ is a constant value, obtained from the range $[0, 1]$. The following algorithm outline shown in Algorithm 5 demonstrates the above explanation.

Algorithm 5: Flower pollination algorithm [17]

Objective min or max $f(x), x = (x_1, x_2, \ldots, x_D)$
Initialise a population of n flowers/pollen gametes with random solutions
Find the best solution g_* in the initial population
Define a switch probability $p \in [0, 1]$
while $(t < MaxGeneration)$
  for $i = 1 : n$ $(all\ n\ flowers\ in\ the\ population)$
    if $rand < p$
      Draw a (d-dimensional) step vector L which obey a Levy distribution
      Global pollination via $x_i^{t+1} = x_i^t + L(g_* - x_i^t)$
    Else
      Draw $\in$ from a uniform distribution in [0, 1]
      Randomly choose j and k among all solutions
      Do local pollination via $x_i^{t+1} = x_i^t + \in (x_j^t - x_k^t)$
    end if
    Evaluate new solutions
    If new solutions are better, update them in the population
  End for
  Find the current best solution g_*
End while

Symbiotic organisms search algorithm

The SOS algorithm is inspired by the interactive behaviour between different organisms in nature. Organisms normally live with each other and they rely on each other for survival and feeding. This kind of relationship is called symbiosis [12]. There are two kinds of symbiotic relationship, namely, obligate relationship and facultative relationship [12]. Obligate relationship refers to the relationship where two organisms live with each other and depend on each other for survival, while facultative relationship refers to the relationship where two organisms live with each other, but do not depend on each other for survival.

The SOS algorithm was designed to handle problems in a continuous space, and it starts with an initial population, also known as the ecosystem. First, different organisms are randomly generated in the ecosystem, where each organism refers to the problem at hand. Each organism is attached to a specific fitness value, which mirrors the amount of change to the defined objective function. Furthermore, a new set of solutions are generated based on the interactive behaviour between two organisms in the population or ecosystem. There are different kinds of symbiotic relationships found in nature, but the most popular relationships include parasitism, mutualism and commensalism. Also, there are three major phases in SOS and these phases are designed to mimic parasitism, mutualism and commensalism. In the parasitic phase, interactions between organisms benefit one party but the other is disadvantaged. Also, in the mutualism phase, interaction benefits both parties, while in commensalism, interactions benefit only one party but does not harm or affect the other party. Each organism randomly interacts with the other organisms in the ecosystem through all the aforementioned phases. These phases are briefly explained next.

As aforementioned, mutualism describes the interaction between two different organisms that benefits both organisms. The two organisms (x_i and x_j) interact with each other with the goal of increasing their chances of survival in the ecosystem [5, 12, 47]. As shown in Eqs. (22) and (23), new solutions for the two organisms (x_i and x_j) are calculated based on the interactions between both of them.

$$X_{i\text{new}} = X_i + \text{rand}(0, 1) * (X_{\text{best}} - \text{MV} * \text{BF}_1) \tag{22}$$

$$X_{j\text{new}} = X_j + \text{rand}(0, 1) * (X_{\text{best}} - \text{MV} * \text{BF}_2) \tag{23}$$

$$\text{MV} = \frac{X_i + X_j}{2} \tag{24}$$

where $\text{rand}(0, 1)$ refers to random numbers between 0 and 1, BF_1 and BF_2 represent benefit factors for the two interacting organisms. The benefit factor represents the level of benefit for each organism—partial or full. MV refers to mutual vector, which represents the relationship features between the two interacting organisms.

Commensalism describes the relationship between two organisms, where one of the organism benefits and the other is not harmed or affected. During interaction, a new solution for the benefitting candidate (X_i) is calculated using Eq. (25).

$$X_{i\text{new}} = X_i + \text{rand}(-1, 1) * (X_{\text{best}} - X_J) \tag{25}$$

where $\text{rand}(-1, 1)$ refers to random numbers between -1 and 1, X_{best} refers to the best organism in the ecosystem.

Parasitism describes the relationship between two organisms, where one of the organism benefits and the other is disadvantaged. In this relationship, the parasite organism (X_i) is given a role, by creating a parasite vector. The parasite vector is created in the ecosystem by duplicating the parasite organism (X_i), and then modifying a randomly selected dimension with the aid of a random number. Further, the host organism (X_j) is randomly selected, which serves as the host to the parasite organism (X_i). The fitness value for the host and parasite organisms are both calculated, and if the fitness value of the host organism is better, then the parasite organism will be eliminated from the ecosystem, otherwise, the host organism will be eliminated. A pseudocode for the SOS algorithm is presented in Algorithm 6.

Algorithm 6: Symbiotic Organisms Search algorithm

Begin SOS
  Randomly Initialise population
  Evaluate the fitness of initial population
  Retain the best solution
  While (termination point is not reached)
    Mutualism phase
    Commensalism phase
    Parasitism phase
    Evaluate fitness of the population
    Update the best solution
  End while
  return best solution
End SOS

Particle swarm optimisation

The PSO is a swarm intelligence-based metaheuristic inspired by the schooling and flocking patterns exhibited by birds and fishes. PSO has three basic behaviours, namely, separation, alignment and cohesion [48]. Separation is used to describe the behaviour that makes an agent to move away from other agents, and alignment is used to describe the behaviour that makes agents to move in the direction of other agents [49]. Cohesion describes the behaviour that makes agents to move towards the average position of other agents in the population [49].

PSO algorithm starts by assigning random positions to each particle of the swarm. Furthermore, the fitness value for each particle is evaluated and then the individual best, velocity and position for each particle are updated accordingly. Also, at each iteration, the particle with the best fitness value is identified and its fitness value and position is stored as the global best. Further, the fitness value for the updated particle is calculated and the individual best, global best, velocity and position of each particle are updated for the second time. This process continues until a user-defined termination value is reached. Finally, the best solution—the solution with the highest fitness value, is selected. The following formulae are used to describe the behaviour of PSO [50]:

$$v_{id}^{t+1} = v_{id}^{t} + c_1 * \text{rand}(0,1) * \left(p_{id}^{t} - x_{id}^{t}\right) + c_2 * \text{rand}(0,1) * \left(p_{gd}^{t} - x_{id}^{t}\right) \tag{26}$$

$$x_{id}^{t+1} = x_{id}^{t} + v_{id}^{t+1} \tag{27}$$

where x_{id}^{t} and v_{id}^{t} represent the position and velocity of each particle, respectively. The parameter d represents the dimension of the problem, i is the index of the each particle and t represents the number of iterations. The parameters c_1 and c_2 are learning factors that control the acceleration of each particle when moving towards the best positions in the population. p_i refers to the current best position achieved by the ith particle and p_g refers to the current best position identified by the neighbours of the ith particle. $\text{rand}(0,1)$ refers to random values between 0 and 1, p_{id}^{t} refers to the current best position of particle i at iteration t and p_{gd}^{t} refers to the population global best particle. PSO can be implemented as outlined in Algorithm 7.

Algorithm 7: Particle Swarm Optimisation [10]

```
Begin PSO
  Initialise population
  Evaluate the fitness of initial population
  Retain the best solution and set as g_best
  While (termination condition is not reached)
      For each particle x with position p_i
          Calculate fitness value
          If fitness value is greater than the current best fitness value (p_best)
              Set current value as p_best
          End if
      End for
      Select the particle with the overall best fitness value and set as g_best
      For each particle
          Calculate particle velocity
          Update position of particle
      End for
  End while
End PSO
```

Differential evolution

Differential evolution is a population-based stochastic real-parameter optimisation algorithm that performs similarly to genetic algorithm. The main difference between DE and GA is in their method of selection. Unlike GA, all solutions in DE has equal probability of being selected as parents. Each child vector is passed through mutation and crossover stages before their fitness value is calculated. Afterwards, the fitness value of the child vector is compared to the fitness value of the parent vector and the fittest vector is retained in the population [16]. DE uses the following properties to generate a new population at different iterations, viz., target vector, mutant vector and trail vector. The target vector contains the solution to the optimisation problem, while the trail vector refers to the resultant vector that is produced after crossover has been performed between the target vector and the mutant vector. Similar to GA, DE employs the following operators, viz., mutation, crossover and selection. DE depends on mutation operation to search for better solutions; it uses the selection operator to tailor the search towards the regions with better solutions [16]. More detailed information on the operators is provided next.

In the mutation stage, a mutant vector is generated for all the target vectors in the solution space according to Eq. (28) [51]:

$$v_{i,G+1} = x_{r1,G} + F\left(x_{r2,G} - x_{r3,G}\right) \tag{28}$$

where F refers to a scaling vector, between the range of 0 and 1, x_{r1}, x_{r3}, x_{r2} are randomly chosen solution vectors which satisfy the following conditions:

$$x_{r1}, x_{r3}, x_{r2} | r_1 \neq r_2 \neq r_3 \neq i \tag{29}$$

where i represent the index of the solution in the current iteration.

Furthermore, as shown in Eq. (30), in crossover, a trail vector is produced by combining the parent vector with a mutated vector [52].

$$u_{i,G+1} = \begin{cases} v_{i,G+1} & \text{if rand}_j \leq \text{CR} \\ x_{i,G} & \text{if rand}_j > \text{CR} \end{cases} \tag{30}$$

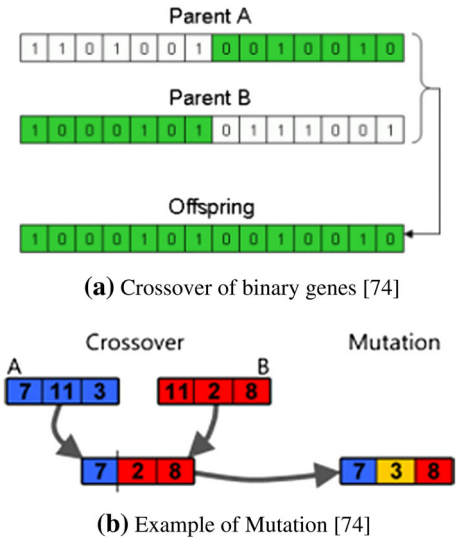
where CR is a constant for crossover, rand_j is a uniform random number in the interval [0,1], and j represents each generated random number in the resulting array. More information about DE is provided in [51]. The details of this algorithm are as shown in Algorithm 8.

Algorithm 8: Differential Evolution

```
Begin DE
  Generate initial population
  Evaluate fitness of initial population
  Retain the best solution
  While (termination condition is not met)
    for each Solution x_i in population
        Generate new solution s_i
        if Fitness(s_i) >= Fitness(x_i)
          Retain s_i in population
        else
          Retain x_i in population
        end if
    end
    Evaluate fitness of the new population
    Update the best solution
  End while
  Return best solution
End DE
```

Genetic algorithm

Genetic algorithm is a population-based metaheuristic inspired by the fascinating process of natural selection. GA was designed based on the mechanism of natural selection, called "survival of the fittest", and it relies on the following NI-based operators to produce good quality solutions to optimisation problems, viz., crossover, reproduction, mutation and selection. A typical example of crossover and mutation is shown in Fig. 6. In GA, a population of individual solutions to certain optimisation problems are usually evolved towards improved solutions. The evolution normally begins by randomly generating an initial population, where each individual in the population consists of a set of chromosomes (strings of 0 s and 1 s), which can be mutated and altered. Also, the fitness of each candidate in the randomly generated population is evaluated by calculating their respective fitness value using a user-defined fitness function. Furthermore, individuals with



(a) Crossover of binary genes [74]

(b) Example of Mutation [74]

Fig. 6 GA crossover and mutation processes

high fitness value are selected and their respective genomes are recombined and mutated (using crossover and mutation operators) to produce a new generation. The crossover operator is used to combine the best chromosomes and to move the best genes to the next generation, while mutation is used to change some genes in each chromosomes with the aim of maintaining diversity from one generation to the other [16]. The selection operator is used to improve the fitness of each population by discarding bad individuals and preserving individuals with good fitness value. The fitness for each chromosome in each new population determines whether they will be preserved to the next generation which is determined by their capacity to reproduce. The algorithm continues by generating new individuals and repetitively applying the aforementioned operators to the newly generated solutions, until a user-defined number of iterations or generations have been reached or until a certain fitness threshold has been reached. Pseudocode for GA is shown in Algorithm 9. For more information, readers are directed to [9, 53].

Algorithm 9: Genetic Algorithm

```
Begin GA
  Randomly generate initial population
  Evaluate fitness of initial population
  Retain best solution
  do
    Calculate fitness value for each candidate in population
    Perform Selection
    Perform Crossover on candidates with high fitness value
    Perform Mutation on candidates with high fitness value
    Update best solution
  while (termination threshold is not met)
  return best solution
End GA
```

Artificial bee colony algorithm

The ABC algorithm is a recently proposed swarm intelligence-based algorithm, introduced by Dervis Karaboga in 2005 [14]. The algorithm is based on the intelligent foraging behaviour of honey bees. In ABC algorithm, artificial agents comprise of three types, namely, the employed bees, the onlooker bees, and the scout bees. Each of these bees plays different roles in the algorithm execution. The ABC algorithm can be divided into some steps as outlined and explained below, for further reading see [14, 16]:

Stage 1 As shown in Eq. (31), in this stage, all the vectors, $\overrightarrow{x_i}$, in the population are initialised to random values by different assigned scout bees and control parameters

$$x_i = l_i + \text{rand}(0,1) * (u_i - l_i) \tag{31}$$

where u_i and l_i refers to the upper bound and lower bound of x_i, respectively.

Stage 2 In this stage, a new food source is randomly selected and added to the population of food sources as shown in Eq. (32). Afterwards, the fitness value of the new food source is evaluated, and either x_i or v_i is selected in a greedy manner, as shown in Eq. (33).

$$v_i = x_i + \emptyset_i(x_i - x_j) \tag{32}$$

where x_j refers to the added food source and $\emptyset_i$ refers to a random number selected from $[-a, a]$

$$fit_i(\overrightarrow{x_i}) = \begin{cases} \dfrac{1}{1+f_i(\overrightarrow{x_i})} & \text{if } f_i(\overrightarrow{x_i}) \geq 0 \\ 1+\text{abs}(f_i(\overrightarrow{x_i})) & \text{if } f_i(\overrightarrow{x_i}) < 0 \end{cases} \tag{33}$$

Algorithm 10: Artificial Bee Colony Algorithm

Begin ABC
  do
    Randomly generate initial population using equation (31)
    Evaluate the fitness of initial population
    Retain the best solution
    Start Employed Bee Search ()
      For each food source $\boldsymbol{x_i}$ in population
        Create a new candidate solution ($\boldsymbol{x_j}$) using updating strategy for employed solution
        Evaluate the fitness for the new solution
        Compare fitness of $\boldsymbol{x_i}$ to $\boldsymbol{x_j}$
        Retain $\boldsymbol{x_i}$ if its fitness is better than $\boldsymbol{x_j}$, otherwise replace $\boldsymbol{x_i}$ with $\boldsymbol{x_j}$ using equation (33)
      End for
    End
    Calculate the probability for each solution and select possible solutions using equation (34)
    Start Onlooker Bee Search ()
      For each food source $\boldsymbol{x_i}$ in selected possible solutions
        Create a new candidate solution ($\boldsymbol{x_j}$) using an updating strategy for onlooker solution
        Evaluate the fitness for the new solution
        Compare fitness of $\boldsymbol{x_i}$ to $\boldsymbol{x_j}$
        Retain $\boldsymbol{x_i}$ if its fitness is better than $\boldsymbol{x_j}$, otherwise replace $\boldsymbol{x_i}$ with $\boldsymbol{x_j}$ using (33)
      End for
    End
    Start Scout Bee Search ()
      Check new solutions
      Search for abandoned food sources and replace with new solutions
      Evaluate fitness of the replaced solutions
    End
    Update best solution
  While (termination threshold is not met)
  return best solution
End ABC

Stage 3 Food sources chosen by onlooker bees depend on the information provided by the employed bees and the fitness value of the employed bees. As shown in Eq. (34), in this stage, the probability p_i is measured by:

$$p_i = \frac{\text{fit}_i\left(\overrightarrow{x_i}\right)}{\sum_{i=1}^{N} \text{fit}_i\left(\overrightarrow{x_i}\right)} \tag{34}$$

where N is the population size.

Stage 4 In this stage, the highest fitness value and its corresponding position is stored in memory. Afterwards, the algorithm goes back to stage 2 and continues until a user-defined termination value is reached. The Pseudocode for ABC algorithm explained above is presented in Algorithm 10 below.

Bee algorithm

The Bee algorithm is based on the foraging behaviour of real life honeybees. In search for food, a bee colony deploys some members of its colony (called scout bees) to different random locations of their surroundings [54, 55]. Specifically, scout bees search for flower patches in areas with high nectar content. They also search in areas where nectar contains high sugar levels, and in areas where nectar can be easily extracted [19]. After identifying potential areas, the scout bees extract some nectar from the various identified areas and return back to their hive. Afterwards, each of the bees deposit the collected nectar and bees with high-quality nectar perform a "waggle dance", which consequently communicates three major sources of information about the location of the discovered nectar to the remaining bees in the hive [19]. The information includes the direction of the discovered flower, the distance between the flower and the hive and the quality of the nectar [19]. After the waggle dance, the dancer bee will lead some recruited members of its colony to the location of the food source. The number of recruited members depends on the quality of nectar. Furthermore, when a recruited bee returns to the hive, it also performs waggle dance with the aim of leading the remaining bees to the food location. This process enables the bee colony to collect the maximum amount of food at low harvesting cost. Food sources with high-quality nectar always attract the largest number of bees.

The BeeA was designed to identify the best position in the search space of given optimisation problems. Each solution in the search space is a potential solution to the problem at hand, and is considered to be a food source. Initially, different solutions (or points) are randomly generated in the search space and artificial scout bees are deployed to randomly search some portion of the generated solutions. Subsequently, the scout bees report the fitness values for each of the generated solutions, and more artificial bees are then employed to examine the search space of the solutions with the highest fitness value. The goal of this search is to identify the global best solution as defined by the objective function. The following algorithm outline shown in Algorithm 11 reflects the above explanation.

Algorithm 11: The Bees Algorithm

```
Begin BeeA
    Randomly generate initial solutions
    Evaluate the fitness value of the initial solutions
    Retain the best solution
    While (termination threshold is not met)
        Select sites (or portion of solution) for search
        Deploy scout bees to search selected sites
        Evaluate the fitness of the searched solutions
        Select solutions with high fitness value
        Deploy remaining bees to examine locations with low fitness value
        Evaluate the fitness value of the examined solutions
        Update global best
    End while
    Return global best
End BeeA
```

Invasive weed optimisation

Invasive weed optimisation is inspired by the aggressive growth and colonisation of weed plants. Generally, weeds are plants whose invasive or aggressive growth threatens the growth of desired plants. The IWO algorithm was designed using some properties of weed colonisation process, discussed below [18, 56]:

The algorithm begins by randomly dispersing a specific number of seed over a d dimensional solution space. Every dispersed seed grows to become flowering plants and reproduces seeds based on their respective fitness value and the colony's lowest and maximum fitness value. This is to ensure that there is a linear increase in the number of seeds produced by each plant, from a user-defined minimum to a user-defined maximum value. Furthermore, the reproduced seeds are dispersed to random locations in the search space and grow to become new plants. The algorithm is designed to disperse seeds to regions close to their parent plants. The algorithm continues until the colony reaches a specified maximum number of weeds. Afterwards, plants with lower fitness value are eliminated based on the following elimination mechanism:

- Each weed is permitted to reproduce seeds based on their fitness value.
- The reproduced seeds are then dispersed to random locations across the entire search space, which then grow to become new plants or weeds.
- The new weeds are then compared to their parent weeds, and weeds with lower fitness values are

eliminated, while weeds with higher fitness value are retained and allowed to reproduce.

This mechanism permits weeds of lower fitness value to reproduce, since weeds of lower fitness value may also carry better information compared to weeds of higher fitness value. The algorithm is finally terminated after a user-defined number of runs is reached. More information about the algorithm is provided in [18]. The pseudocode for the IWO algorithms is shown in Algorithm 12.

Algorithm 12: Invasive Weed Optimisation Algorithm

```
Begin IWO
  Define parameters, including: highest number of weeds (h) and lowest number of seeds (l)
  Randomly generate initial solution of weeds
  Evaluate the fitness of the initial population
  Retain the global best plant
  While (Colony is less than specified number of weeds)
      Evaluate fitness of the generated solution
      Disperse specific number of seeds over a solution space
      Evaluate fitness of each examined solution
      Reproduce new solutions using the following: solution fitness value, h and l
      Disperse reproduced seeds to random locations in the search space using normal distribution
      if (Maximum number of specified seeds is reached)
          Calculate N weeds to eliminate to maintain the maximum number of specified seed
          Each weed is allowed to reproduce seeds
          Evaluate fitness of the new (or reproduced) solution
          Compare fitness of each new seed to fitness of its parent seed
          Eliminate N weeds with low fitness value
      End if
      Update global best plant
  End while
  Return global best
End IWO
```

Appendix 2: Benchmark of global optimisation test functions

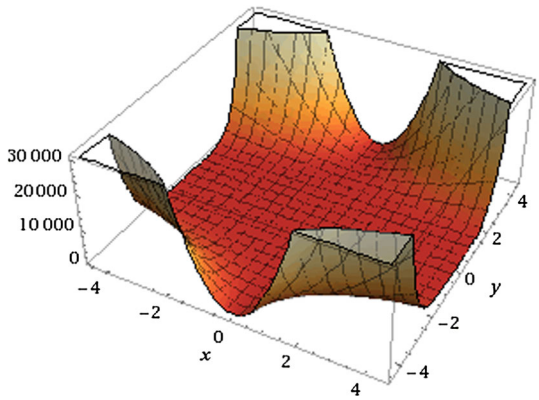
This section presents a brief mathematical description of thirty-six standard benchmark functions, often used by researchers in the global optimisation domain to test and evaluate the effectiveness and efficiency of most global optimisation algorithms. However, the benchmark functions detailed in this paper are only a careful selection of many sparsely available test problems from different sources. The major motivation for this representation is to provide a collection of easily accessible standard benchmark functions, which can be very challenging to collate from a single source. Therefore, it is assumed that the 36 benchmark test functions compilation described in this paper will eventually be adopted as a subset of standard test problems for evaluating metaheuristic algorithms in continuous optimisation and other related optimisation application domains. Note that in each of the figures given in this section, $x = x_1, y = x_2$.

1. **Beale function**: $f(x_1, x_2) = (1.5 - x_1 + x_1x_2)^2 + (2.25 - x_1 + x_1x_2^2)^2 + (2.625 - x_1 + x_1x_2^3)^2$

   Characteristics: continuous, nonconvex, multimodal, defined on a 2-dimensional space.

   Search space: $\vec{x} = (x_1, x_2) \in [-4.5, 4.5]$.

   Global minimum: $f(\vec{x}^*) = 0$ at $\vec{x}^* = (3, 0.5)$.

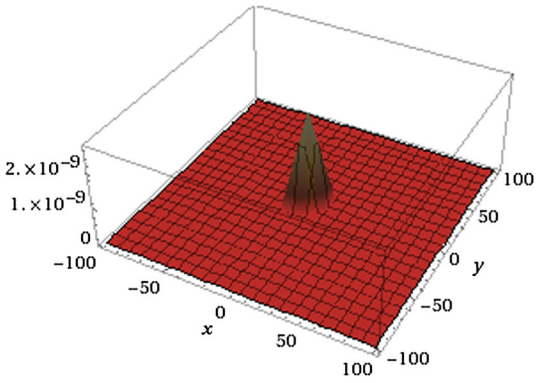


2. **Easom function**: $f(x_1, x_2) = -\cos(x_1)\cos(x_2)\exp(-(x_1 - \pi)^2 - (x_2 - \pi)^2)$

   Characteristics: continuous, differentiable, nonconvex, non-separable, multimodal, defined on a 2-dimensional space.

   Search space: $\vec{x} = (x_1, x_2) \in [-100, 100]$.

   Global minimum: $f(\vec{x}^*) = 1$ at $\vec{x}^* = (\pi, \pi)$.

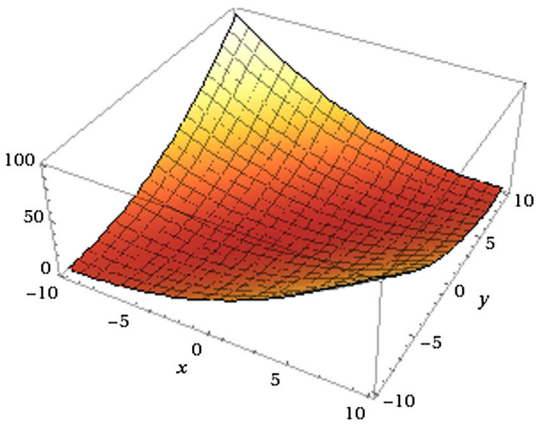


3. **Matyas function**: $f(x_1, x_2) = 0.26(x_1^2 + x_2^2) - 0.48x_1x_2$

   Characteristics: continuous, differentiable, non-separable, convex, unimodal and defined on a 2-dimensional space.

   Search space: $\vec{x} = (x_1, x_2) \in [-10, 10]$.

   Global minimum: $f(\vec{x}^*) = 0$ at $\vec{x}^* = (0, 0)$.

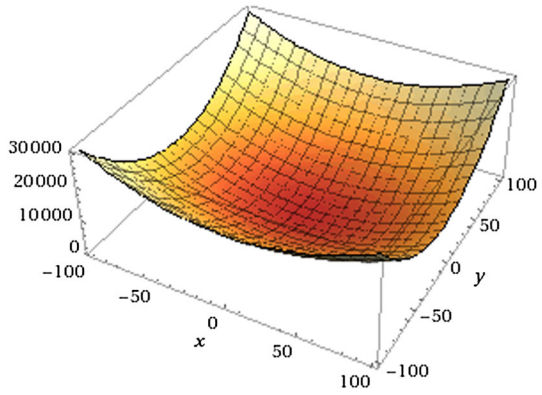


4. **Bohachevsky 1 function**: $f(x_1, x_2) = x_1^2 + 2x_2^2 - 0.3\cos(3\pi x_1) - 0.4\cos(4\pi x_2) + 0.7$

   Characteristics: continuous, differentiable, separable, convex, unimodal and defined on a 2-dimensional space.

   Search space: $\vec{x} = (x_1, x_2) \in [-100, 100]$.

   Global minimum: $f(\vec{x}^*) = 0$ at $\vec{x}^* = (0, 0)$.

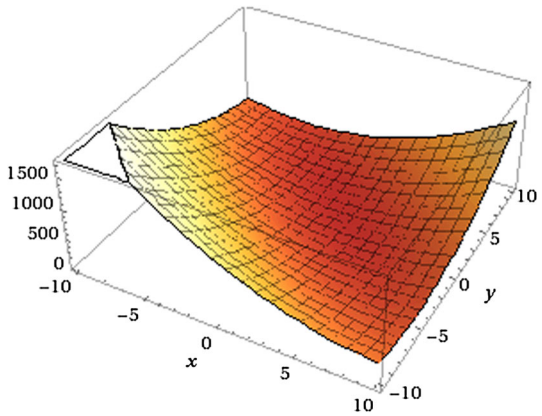


5. **Booth function**: $f(x_1, x_2) = (x_1 + 2x_2 - 7)^2 + (2x_1 + x_2 - 5)^2$

   Characteristics: continuous, differentiable, non-separable, convex, unimodal and defined on a 2-dimensional space.

   Search space: $\vec{x} = (x_1, x_2) \in [-10, 10]$.

   Global minimum: $f(\vec{x}^*) = 0$ at $\vec{x}^* = (1, 3)$.

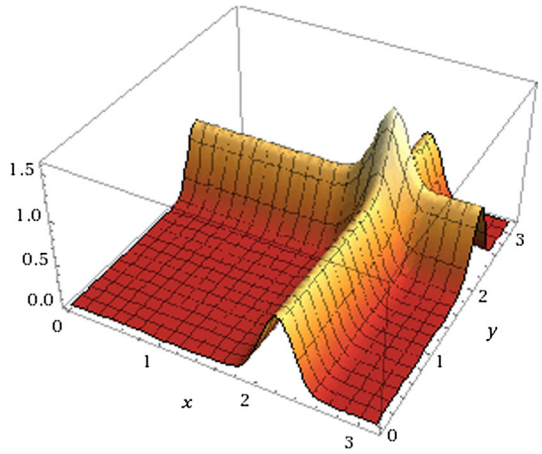


6. **Michalewics 2 function**: $f(x) = -\sum_{i=1}^{2} \sin(x_i) \left[\sin\left(ix_i^2/\pi\right)\right]^{2m}$

   Characteristics: continuous, differentiable, separable, multimodal and defined on a 2-dimensional space. m is usually taken to be equal to 10.

   Search space: $\vec{x} = (x_1, x_2) \in [0, \pi]$.

   Global minimum: $f(\vec{x}^*) = -1.8013$ at $\vec{x}^* = (2.20, 1.57)$

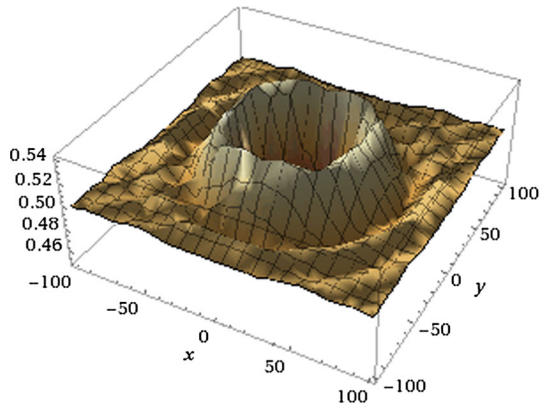


7. **Schaffer function**: $f(x_1, x_2) = 0.5 + \frac{\sin^2(x_1^2+x_2^2)^2 - 0.5}{(1+0.001(x_1^2+x_2^2))^2}$

   Characteristics: continuous, differentiable, non-separable, nonconvex, unimodal and defined on a 2-dimensional space.

   Search space: $\vec{x} = (x_1, x_2) \in [-100, 100]$.

   Global minimum: $f(\vec{x}^*) = 0$ at $\vec{x}^* = (0, 0)$.

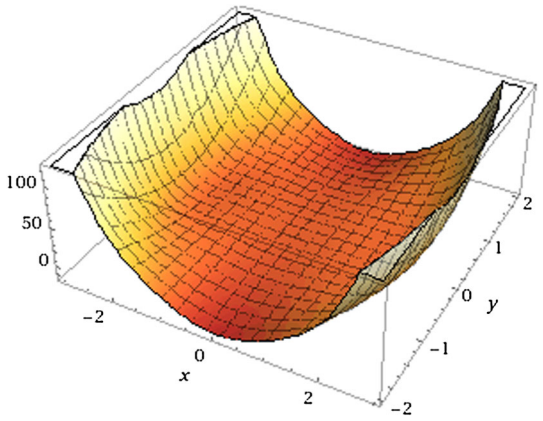


8. **Six hump camel back**: $f(x_1, x_2) = (4 - 2.1x_1^2 + (x_1^4/3))x_1^2 + x_1x_2 + (-4 + 4x_1^2)x_2^2$

   Characteristics: continuous, differentiable, non-separable, nonconvex, unimodal and defined on a 2-dimensional space.

   Search space: $x_1 \in [-3, 3], x_2 \in [-2, 2]$.

   Global minimum: $f(\vec{x}^*) = -1.0316$ at $\vec{x}^* = (x_1, x_2) = (-0.0898, 0.7126)$ and $\vec{x}^* = (x_1, x_2) = (0.0898, -0.7126)$.

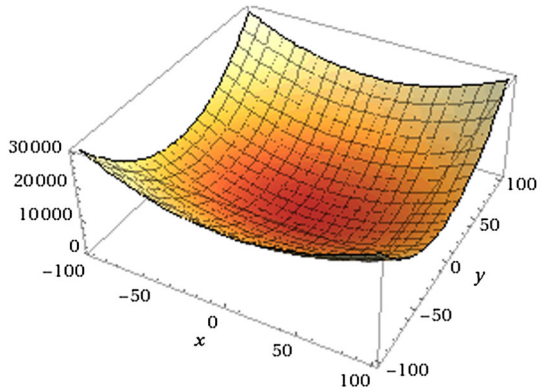


9. **Bohachevsky 2 function**: $f(x_1, x_2) = x_1^2 + 2x_2^2 - 0.3\cos(3\pi x_1)\cos(4\pi x_2) + 0.3$

   Characteristics: continuous, differentiable, non-separable, nonconvex, multimodal and defined on a 2-dimensional space.

   Search space: $\vec{x} = (x_1, x_2) \in [-100, 100]$.

   Global minimum: $f(\vec{x}^*) = 0$ at $\vec{x}^* = (0, 0)$.

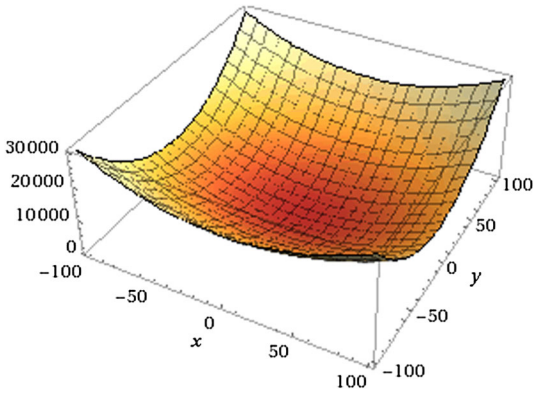


10. **Bohachevsky 3 function**: $f(x_1, x_2) = x_1^2 + 2x_2^2 - 0.3\cos(3\pi x_1 + 4\pi x_2) + 0.3$

    Characteristics: continuous, differentiable, non-separable, nonconvex, multimodal and defined on a 2-dimensional space.

    Search space: $\vec{x} = (x_1, x_2) \in [-100, 100]$.

    Global minimum: $f(\vec{x}^*) = 0$ at $\vec{x}^* = (0, 0)$.

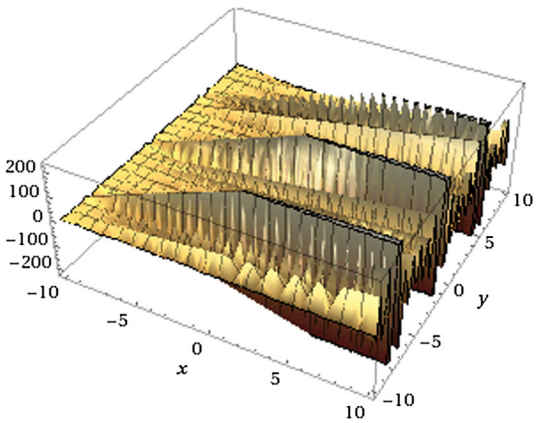


11. **Shubert function**: $f(x_1, x_2) = \left(\sum_{i=1}^{5} i\cos(i+1)x_1 + i\right)\left(\sum_{i=1}^{5} i\cos((i+1)x_2 + i)\right)$

Characteristics: continuous, differentiable, non-separable, nonconvex, multimodal and defined on a 2-dimensional space.

Search space: $\vec{x}^* = (x_1, x_2) \in [-10, 10]$.

Global minimum: the function has 18 global minima and $f(\vec{x}^*) \approx -186.7309$.



12. **Colville function**: $f(\vec{x}) = 100(x_1^2 - x_2)^2 + (x_1 - 1)^2 + (x_3 - 1)^2 + 90(x_3^2 - x_4)^2 + 10.1((x_2 - 1)^2 + (x_4 - 1)^2) + 19.8(x_2 - 1)(x_4 - 1)$

where $\vec{x} = (x_1, x_2, x_3, x_4)$

Characteristics: continuous, differentiable, non-separable, nonconvex, multimodal and defined on a 4-dimensional space.

Search space: $\vec{x} \in [-10, 10]$.

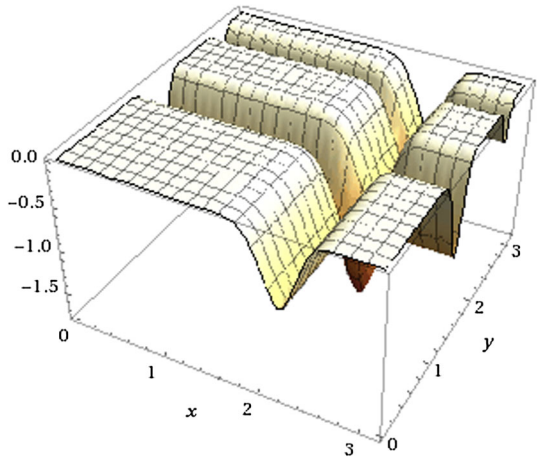
Global minimum:$f(\vec{x}^*) = 0$ at $\vec{x}^* = (1, 1, 1, 1)$.

13. **Michalewicz 5 function**: $f(x) = -\sum_{i=1}^{n} \sin(x_i)\left[\sin\left(ix_i^2/\pi\right)\right]^{2m}$

Characteristics: continuous, separable, multimodal and defined on a 5-dimensional space.

Search space: $\vec{x} = (x_1, x_2, x_3, x_4, x_5) \in [0, \pi]$.

Global minimum: $f(\vec{x}^*) = -4.687658$.

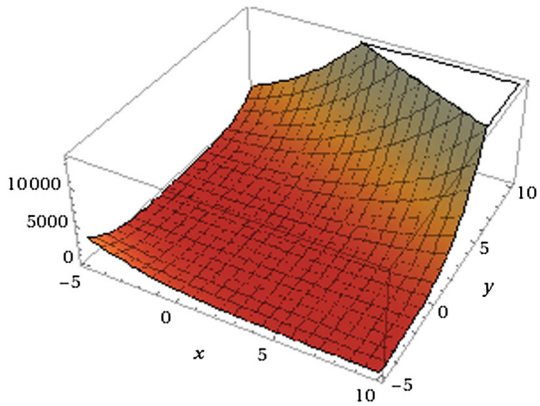


14. **Zakharov function**: $f(x) = \sum_{i=1}^{10} x_i^2 + \left(\sum_{i=1}^{10} 0.5ix_i\right)^2 + \left(\sum_{i=1}^{10} 0.5ix_i\right)^4$

Characteristics: continuous, non-separable, convex, unimodal and defined on a 10-dimensional space.

Search space: $\vec{x} \in [-5, 10]$.

Global minimum: $f(\vec{x}^*) = 0$ at $\vec{x}^* = (0, \ldots, 0)$.



15. **Michalewicz 10 function**: $f(\vec{x}) = -\sum_{i=1}^{n} \sin(x_i)\left[\sin\left(ix_i^2/\pi\right)\right]^{2m}$

Characteristics: continuous, separable, multimodal and defined on a 10-dimensional space.

Search space: $\vec{x} \in [0, \pi]$.

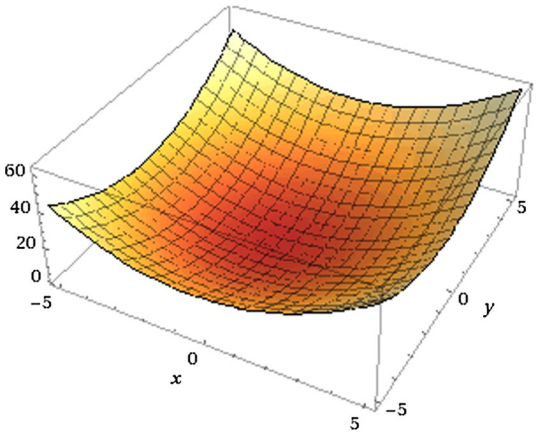
Global minimum: $f(\vec{x}^*) = -9.66015$.

16. **Step function**: $f(\vec{x}) = \sum_{i=1}^{30} (x_i + 0.5)^2$

Characteristics: continuous, separable, unimodal and defined on an n-dimensional space.

Search space: $\vec{x} \in [-5.12, 5.12]$.

Global minimum: $f(\vec{x}^*) = 0$.

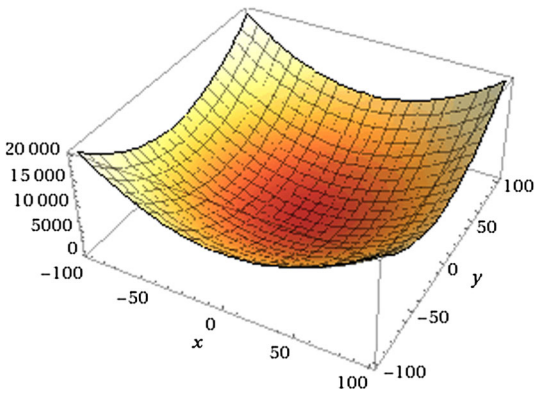


17. **Sphere function**: $f(\vec{x}) = \sum_{i=1}^{30} x_i^2$

Characteristics: continuous, differentiable, separable, convex, unimodal and defined on an n-dimensional space.

Search space: $\vec{x} \in [-100, 100]$.

Global minimum: $f(\vec{x}^*) = 0$ at $\vec{x}^* = (0, \ldots, 0)$.

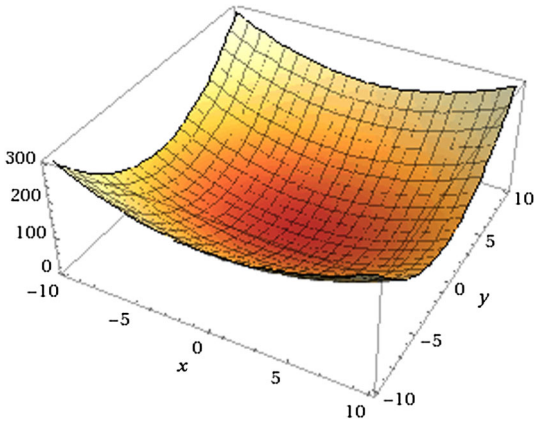


18. **Sum squares function**: $f(\vec{x}) = \sum_{i=1}^{30} ix_i^2$

Characteristics: continuous, differentiable, separable, convex, unimodal and defined on an n-dimensional space.

Search space: $\vec{x} \in [-10, 10]$.

Global minimum: $f(\vec{x}^*) = 0$ at $\vec{x}^* = (0, \ldots, 0)$.

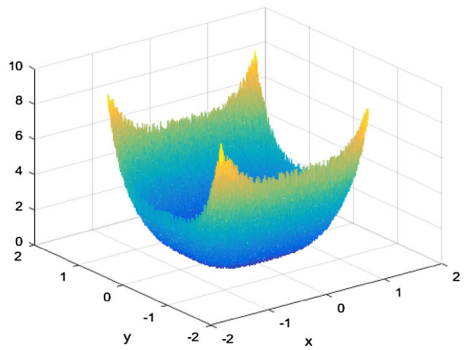


19. **Quartic function**: $f(\vec{x}) = \sum_{i=1}^{30} ix_i^4 + \text{random}[0, 1)$

Characteristics: continuous, differentiable, separable, nonconvex, unimodal and defined on an n-dimensional space.

Search space: $\vec{x} \in [-1.28, 1.28]$.

Global minimum: $f(\vec{x}^*) = 0 + \text{random noise}$ at $\vec{x}^* = (0, \ldots, 0)$.

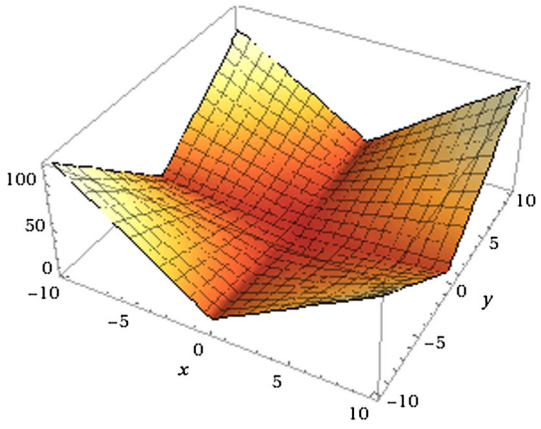


20. **Schwefel 2.22 function**: $f(\vec{x}) = \sum_{i=1}^{30} |x_i| + \prod_{i=1}^{30} |x_i|$

Characteristics: continuous, non-differentiable, separable, convex, unimodal and defined on a 30-dimensional space.

Search space: $\vec{x} \in [-10, 10]$.

Global minimum: $f(\vec{x}^*) = 0$ at $\vec{x}^* = (0, \ldots, 0)$.

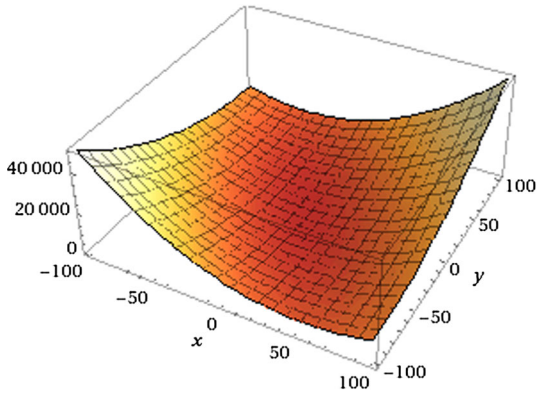


21. **Schwefel 1.2 function**: $f(\vec{x}) = \sum_{i=1}^{30} \left(\sum_{j=1}^{i} x_j \right)^2$

Characteristics: continuous, differentiable, non-separable, unimodal and defined on an n-dimensional space.

Search space: $\vec{x} \in [-100, 100]$.

Global minimum: $f(\vec{x}^*) = 0$ at $\vec{x}^* = (0, \ldots, 0)$.

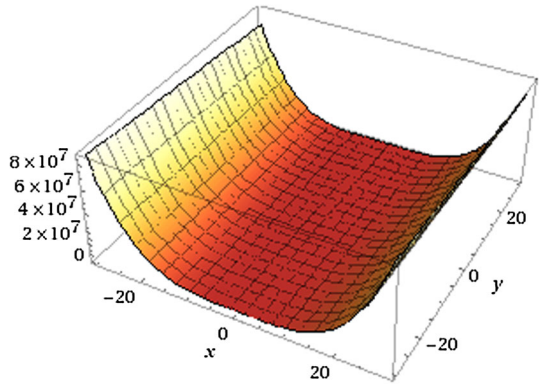


22. **Rosenbrock function**: $f(\vec{x}) = \sum_{i=1}^{n-1} [100(x_{i+1} - x_i^2)^2 + (x_i - 1)^2]$ where $n = 30$.

Characteristics: continuous, differentiable, convex, non-separable, unimodal and defined on an n-dimensional space.

Search space: $\vec{x} \in [-30, 30]$.

Global minimum: $f(\vec{x}^*) = 0$ at $\vec{x}^* = 1$.

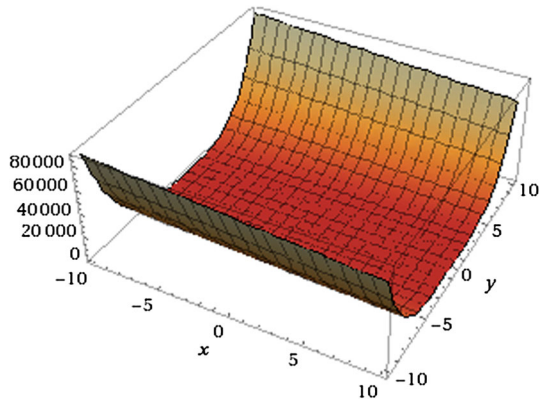


23. **Dixon Price function**: $f(\vec{x}) = (x_1 - 1)^2 + \sum_{i=2}^{n} i(2x_i^2 - x_{i-1})^2$ where $n = 30$.

Characteristics: continuous, differentiable, non-separable, unimodal and defined on an n-dimensional space.

Search space: $\vec{x} \in [-10, 10]$.

Global minimum: $f(\vec{x}^*) = 0$ at $\vec{x}_i^* = 2^{-\frac{2^i - 2}{2^i}}$ for $i = 1, \ldots 30$.

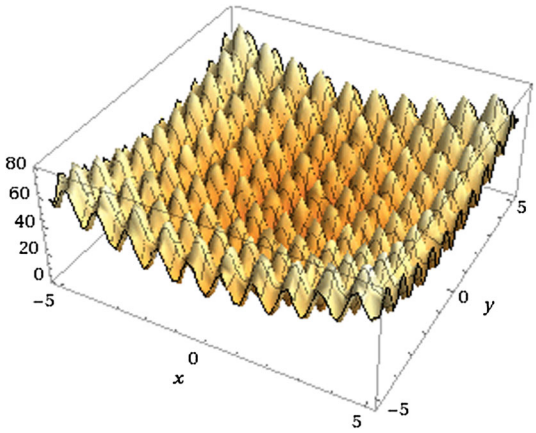


24. **Rastrigin function**: $f(\vec{x}) = 10n + \sum_{i=1}^{n} [x_i^2 - 10\cos(2\pi x_i)]$ where $n = 30$.

Characteristics: continuous, differentiable, convex, separable, multimodal and defined on an n-dimensional space.

Search space: $\vec{x} \in [-5.12, 5.12]$.

Global minimum: $f(\vec{x}^*) = 0$ at $\vec{x}^* = 0$.

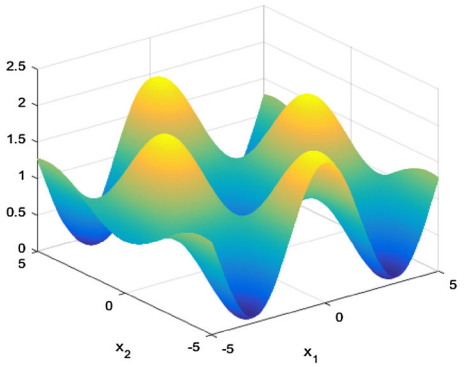


25. **Griewank function**: $f(\vec{x}) = \frac{1}{4000}\sum_{i=1}^{n}(x_i - 100)^2 - \prod_{i=1}^{n}\cos\left(\frac{x_i - 100}{\sqrt{i}}\right) + 1$, where $n = 30$.

Characteristics: continuous, nonconvex, non-separable, unimodal and defined on an n-dimensional space.

Search space: $\vec{x} \in [-600, 600]$.

Global minimum: $f(\vec{x}^*) = 0$ at $\vec{x}^* = 0$.

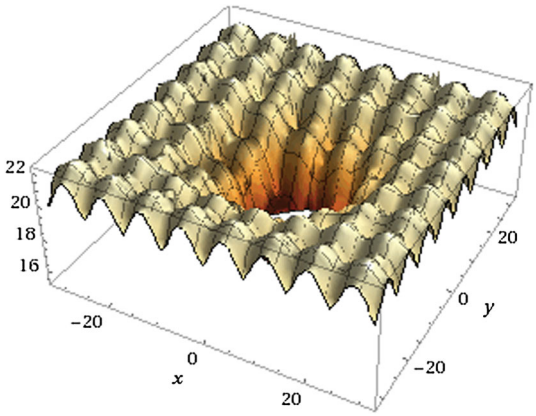


26. **Ackley function**: $f(\vec{x}) = -20\exp\left(-0.2\sqrt{\frac{1}{n}\sum_{i=1}^{n}x_i^2}\right) - \exp\left(\frac{1}{n}\sum_{i=1}^{n}\cos(2\pi x_i)\right) + 20 + \exp(1)$, with $n = 30$.

Characteristics: continuous, nonconvex, non-separable, multimodal and defined on an n-dimensional space.

Search space: $\vec{x} \in [-32, 32]$.

Global minimum: $f(\vec{x}^*) = 0$ at $\vec{x}^* = 0$.

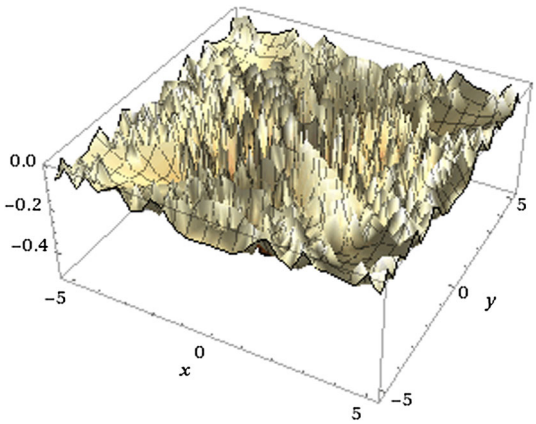


27. **Drop wave function**: $f(\vec{x}) = -\frac{1+\cos\left(12\sqrt{x_1^2+x_2^2}\right)}{0.5\left(x_1^2+x_2^2\right)+2}$, $\vec{x} = (x_1, x_2)$.

Characteristics: continuous, nonconvex, non-separable, multimodal and defined on a 2-dimensional space.

Search space: $\vec{x} \in [-5.2, 5.2]$.

Global minimum: $f(\vec{x}^*) = -1$ at $\vec{x}^* = 0$.



28. **Hartman 3 function**: $f(\vec{x}) = -\sum_{i=1}^{4}A_i \exp\left(-\left(\sum_{j=1}^{3}B_{ij}\left(x_j - C_{ij}\right)^2\right)\right)$,

where $A = (1.0, 1.2, 3.0, 3.2)^T$, $B = \begin{pmatrix} 3.0 & 10 & 30 \\ 0.1 & 10 & 35 \\ 3.0 & 10 & 30 \\ 0.1 & 10 & 35 \end{pmatrix}$, $C = \begin{pmatrix} 3689 & 1170 & 2673 \\ 4699 & 4387 & 7470 \\ 1091 & 8732 & 5547 \\ 381 & 5743 & 8828 \end{pmatrix}$

Characteristics: continuous, differentiable, non-separable, multimodal and defined on a 3-dimensional space.

Search space: $\vec{x} \in [0, 1]$.

Global minimum: $f(\vec{x}^*) \approx -3.862782$ at $\vec{x}^* = (0.1140, 0.556, 0.852)$.

29. **Hartman 6 function**: $f(\vec{x}) = -\sum_{i=1}^{4} A_i \exp\left(-\left(\sum_{j=1}^{6} B_{ij}\left(x_j - C_{ij}\right)^2\right)\right)$,

where $\vec{A} = (1.0, 1.2, 3.0, 3.2)^T$, $\vec{B} =$

$$\begin{pmatrix} 10 & 3 & 17 & 3.5 & 1.7 & 8 \\ 0.05 & 10 & 17 & 0.1 & 8 & 14 \\ 3 & 3.5 & 1.7 & 10 & 17 & 8 \\ 17 & 8 & 0.05 & 10 & 0.1 & 14 \end{pmatrix}, \quad \vec{C} = \begin{pmatrix} 0.1312 & 0.1696 & 0.5569 & 0.0124 & 0.8283 & 0.5586 \\ 0.2329 & 0.4135 & 0.8307 & 0.3736 & 0.1004 & 0.9991 \\ 0.2348 & 0.1451 & 0.3522 & 0.2883 & 0.3047 & 0.6650 \\ 0.4047 & 0.8828 & 0.8732 & 0.5743 & 0.1091 & 0.0381 \end{pmatrix}$$

Characteristics: continuous, differentiable, non-separable, multimodal and defined on a 6-dimensional space.

Search space: $\vec{x} \in [0,1]$.

Global minimum: $f(\vec{x}^*) \approx -3.32236$ at $\vec{x}^* = (0.201690, 0.150011, 0.476874, 0.275332, 0.311652, 0.657301)$.

30. **Shekel 5 function**: $f(\vec{x}) = -\sum_{i=1}^{5} \left(\sum_{j=1}^{4} (x_j - A_{ij})^2 + B_i\right)^{-1}$, where $\vec{A} = \begin{pmatrix} 4 & 4 & 4 & 4 \\ 1 & 1 & 1 & 1 \\ 8 & 8 & 8 & 8 \\ 6 & 6 & 6 & 6 \\ 3 & 7 & 3 & 7 \end{pmatrix}$, $\vec{B} = \begin{pmatrix} 0.1 \\ 0.2 \\ 0.2 \\ 0.4 \\ 0.4 \end{pmatrix}$.

Characteristics: continuous, differentiable, non-separable, multimodal and defined on a 4-dimensional space.

Search space: $\vec{x} \in [0, 10]$.

Global minimum: $f(\vec{x}^*) \approx -10.1499$ at $\vec{x}^* = (4, 4, 4, 4)$.

31. **Shekel 7 function**: $f(\vec{x}) = -\sum_{i=1}^{7} \left(\sum_{j=1}^{4} (x_j - A_{ij})^2 + B_i\right)^{-1}$, where $\vec{A} = \begin{pmatrix} 4 & 4 & 4 & 4 \\ 1 & 1 & 1 & 1 \\ 8 & 8 & 8 & 8 \\ 6 & 6 & 6 & 6 \\ 3 & 7 & 3 & 7 \\ 2 & 9 & 2 & 9 \\ 5 & 5 & 3 & 3 \end{pmatrix}$, $\vec{B} = \begin{pmatrix} 0.1 \\ 0.2 \\ 0.2 \\ 0.4 \\ 0.4 \\ 0.6 \\ 0.3 \end{pmatrix}$.

Characteristics: continuous, differentiable, non-separable, multimodal and defined on a 4-dimensional space.

Search space: $\vec{x} \in [0, 10]$.

Global minimum: $f(\vec{x}^*) \approx -10.3999$ at $\vec{x}^* = (4, 4, 4, 4)$.

32. **Shekel 10 function**: $f(\vec{x}) = -\sum_{i=1}^{10} \left(\sum_{j=1}^{4} (x_j - A_{ij})^2 + B_i\right)^{-1}$, where $\vec{A} = \begin{pmatrix} 4 & 4 & 4 & 4 \\ 1 & 1 & 1 & 1 \\ 8 & 8 & 8 & 8 \\ 6 & 6 & 6 & 6 \\ 3 & 7 & 3 & 7 \\ 2 & 9 & 2 & 9 \\ 5 & 5 & 3 & 3 \\ 8 & 1 & 8 & 1 \\ 6 & 2 & 6 & 2 \\ 7 & 3.6 & 7 & 3.6 \end{pmatrix}$, $\vec{B} = \begin{pmatrix} 0.1 \\ 0.2 \\ 0.2 \\ 0.4 \\ 0.4 \\ 0.6 \\ 0.3 \\ 0.7 \\ 0.5 \\ 0.5 \end{pmatrix}$.

Characteristics: continuous, differentiable, non-separable, multimodal and defined on a 4-dimensional space.

Search space: $\vec{x} \in [0, 10]$.

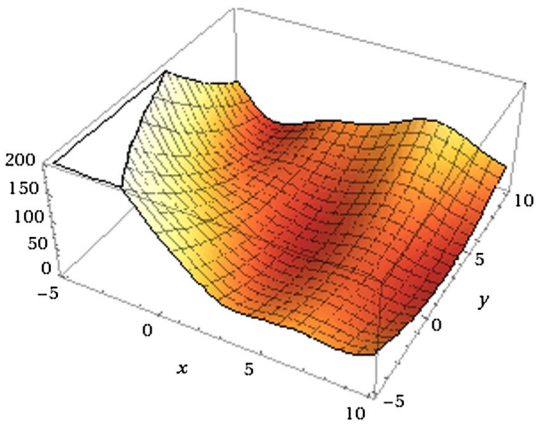
Global minimum: $f(\vec{x}^*) \approx -10.5319$ at $\vec{x}^* = (4, 4, 4, 4)$.

33. **Branin function**: $f(\vec{x}) = \left(x_2 - \frac{5.1x_1^2}{4\pi^2} + \frac{5x_1}{\pi} - 6\right)^2 + 10\left(1 - \frac{1}{8\pi}\right)\cos(x_1) + 10$.

Characteristics: continuous, differentiable, non-separable, multimodal and defined on a 2-dimensional space.

Search space: $\vec{x} \in [-5, 10]$.

Global minimum: $f(\vec{x}^*) \approx 0.3978873$ at $\vec{x}^* = \{(-\pi, 12.275), (\pi, 2.275), (3\pi, 2.425)\}$.

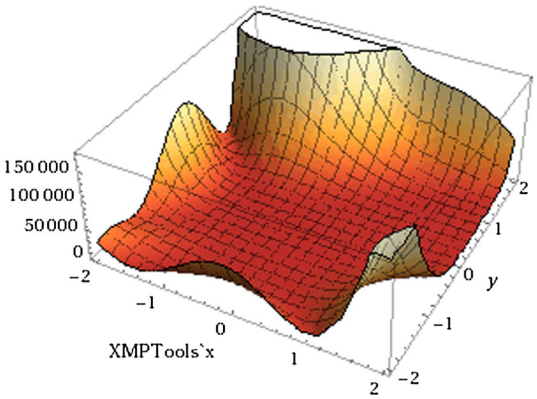


34. **Goldstein-Price function:** $f(\vec{x}) = [1 + (x_1 + x_2 + 1)^2(19 - 14x_1 + 3x_1^2 - 14x_2 + 6x_1x_2 + 3x_2^2)]$ $[30 + (2x_1 - 3x_2)^2(18 - 32x_1 + 12x_1^2 + 48x_2 - 36x_1x_2 + 27x_2^2)]$

Characteristics: continuous, differentiable, non-separable, multimodal and defined on a 2-dimensional space.

Search space: $\vec{x} \in [-2, 2]$.

Global minimum: $f(\vec{x}^*) = 3$ at $\vec{x}^* = (0, -1)$.

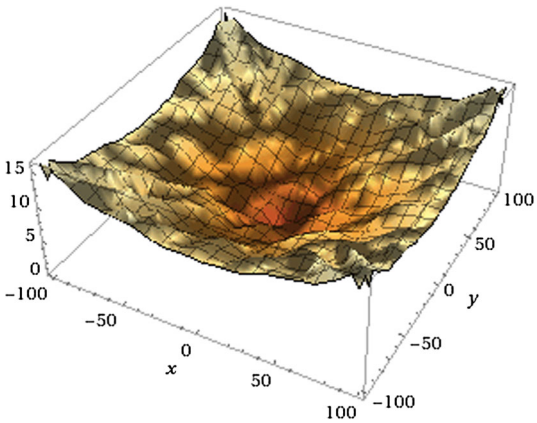


35. **Salomon function (*n* = 5)**: $f(\vec{x}) = 1 - \cos\left(2\pi\sqrt{\sum_{i=1}^{n} x_i^2}\right) + 0.1\sqrt{\sum_{i=1}^{n} x_i^2}$ with $n = 5$.

Characteristics: continuous, differentiable, non-separable, multimodal and defined on an *n*-dimensional space.

Search space: $\vec{x} \in [-100, 100]$.

Global minimum: $f(\vec{x}^*) = 0$ at $\vec{x}^* = 0$.



36. **Salomon function (*n* = 6)**: $f(\vec{x}) = 1 - \cos\left(2\pi\sqrt{\sum_{i=1}^{n} x_i^2}\right) + 0.1\sqrt{\sum_{i=1}^{n} x_i^2}$ with $n = 6$.

Characteristics: continuous, differentiable, non-separable, multimodal and defined on an *n*-dimensional space.

Search space: $\vec{x} \in [-100, 100]$.

Global minimum: $f(\vec{x}^*) = 0$ at $\vec{x}^* = 0$.

References

1. Zhou JX, Zou W (2008) Meshless approximation combined with implicit topology description for optimisation of continua. Struct Multidiscip Optim 36(4):347–353
2. Luo Z, Zhang N, Wang Y, Gao W (2013) Topology optimisation of structures using meshless density variable approximants. Int J Numer Methods Eng 93(4):443–464
3. Lin J, Chen CS, Liu CS, Lu J (2016) Fast simulation of multi-dimensional wave problems by the sparse scheme of the method of fundamental solutions. Comput Math Appl 72(3):555–567
4. Lin J, Reutskiy SY, Lu J (2018) A novel meshless method for fully nonlinear advection–diffusion–reaction problems to model transfer in anisotropic media. Appl Math Comput 339:459–476
5. Fu ZJ, Xi Q, Chen W, Cheng AHD (2018) A boundary-type meshless solver for transient heat conduction analysis of slender functionally graded materials with exponential variations. Comput Math Appl 76:760–773
6. Fister I, JrFister I, Yang X-S, Brest J (2013) A comprehensive review of firefly algorithms. Swarm Evolut Comput 13:34–46
7. Jamil M, Yang XS (2013) A literature survey of benchmark functions for global optimisation problems. Int J Math Model Numer Optim 4(2):150–194
8. Dieterich JM, Hartke B (2012) Empirical review of standard benchmark functions using evolutionary global optimisation. Appl Math 3:1552–1564
9. Holland JH (1992) Genetic algorithms. Sci Am 267(1):66–73
10. Goldbarg EF, Goldbarg MC, de Souza GR (2008) Particle swarm optimisation algorithm for the traveling salesman problem. In: Traveling salesman problem, ed: InTech
11. Dorigo M, Birattari M, Stutzle T (2006) Ant colony optimisation. IEEE Comput Intell Mag 1(4):28–39
12. Cheng M-Y, Prayogo D (2014) Symbiotic organisms search: a new metaheuristic optimisation algorithm. Comput Struct 139:98–112
13. Yang X-S, Deb S (2009) Cuckoo search via Lévy flights. In: 2009. NaBIC 2009. World Congress on nature & biologically inspired computing, pp 210–214
14. Karaboga D, Basturk B (2007) A powerful and efficient algorithm for numerical function optimisation: artificial bee colony (ABC) algorithm. J Glob Optim 39(3):459–471
15. Yang X-S (2010) A new metaheuristic bat-inspired algorithm. In: González JR, Pelta DA, Cruz C, Terrazas G, Krasnogor N (eds) Nature inspired cooperative strategies for optimisation (NICSO 2010). Springer, Berlin, pp 65–74
16. Ab Wahab MN, Nefti-Meziani S, Atyabi A (2015) A comprehensive review of swarm optimisation algorithms. PLoS ONE 10(5):e0122827
17. Yang X-S (2012) Flower pollination algorithm for global optimisation. In: International conference on unconventional computing and natural computation, pp 240–249
18. Mehrabian AR, Lucas C (2006) A novel numerical optimisation algorithm inspired from weed colonization. Ecol Inf 1(4):355–366
19. Pham DT, Castellani M (2009) The bees algorithm: modelling foraging behaviour to solve continuous optimisation problems. Proc Inst Mech Eng Part C J Mech Eng Sci 223(12):2919–2938
20. Dolan ED, Moré JJ (2002) Benchmarking optimisation software with performance profiles. Math Program 91(2):201–213
21. Ma H, Simon D, Fei M, Chen Z (2013) On the equivalences and differences of evolutionary algorithms. Eng Appl Artif Intell 26(10):2397–2407
22. Ma H, Ye S, Simon D, Fei M (2017) Conceptual and numerical comparisons of swarm intelligence optimisation algorithms. Soft Comput 21(11):3081–3100

23. Civicioglu P, Besdok EA (2013) A conceptual comparison of the Cuckoo-search, particle swarm optimisation, differential evolution and artificial bee colony algorithms. Artif Intell Rev 39(4):315–346
24. Soler-Dominguez A, Juan AA, Kizys R (2017) A survey on financial applications of metaheuristics. ACM Comput Surv (CSUR) 50(1):15
25. Yang XS (2010) Engineering optimisation: an introduction with metaheuristic applications. Wiley, New York
26. Ezugwu AE, Prayogo D (2019) Symbiotic organisms search algorithm: theory, recent advances and applications. Expert Syst Appl 119(1):184–209
27. Ali MM, Khompatraporn C, Zabinsky ZB (2005) A numerical evaluation of several stochastic algorithms on selected continuous global optimisation test problems. J Glob Optim 31(4):635–672
28. Yang XS (2010) Firefly algorithm, Lévy flights and global optimization. In: Bramer M, Ellis R, Petridis M (eds) Research and development in intelligent systems, vol XXVI. Springer, London, pp 209–218
29. Yang XS, Karamanoglu M, He X (2014) Flower pollination algorithm: a novel approach for multiobjective optimisation. Eng Optim 46(9):1222–1237
30. Gendreau M, Potvin JY (2010) Handbook of metaheuristics. Springer, New York
31. Amodeo L, Talbi EG, Yalaoui F (eds) (2018) Recent developments in metaheuristics. Springer, Berlin
32. Zhang L, Liu L, Yang X-S, Dai Y (2016) A novel hybrid firefly algorithm for global optimisation. PLoS ONE 11(9):e0163230
33. Ross K (2017) ISM206: metaheuristics. https://classes.soe.ucsc.edu/ism206/Fall05/Lecture12.pdf. Accessed 23 Jan 2018
34. Silberholz J, Golden B (2010) Comparison of metaheuristics. In: Gendreau M, Potvin JY (eds) Handbook of metaheuristics. International series in operations research & management science, vol 146. Springer, Boston
35. Lobo FJ, Lima CF, Michalewicz Z (eds) (2007) Parameter setting in evolutionary algorithms, 54th edn. Springer, Berlin
36. Shi Y, Eberhart RC (1998) Parameter selection in particle swarm optimization. In: Porto VW, Saravanan N, Waagen D, Eiben AE (eds) Evolutionary programming VII. EP 1998. Lecture notes in computer science, vol 1447. Springer, Berlin, Heidelberg
37. Rajabioun R (2011) Cuckoo optimisation algorithm. Appl Soft Comput 11(8):5508–5518
38. Yang XS, Deb S (2014) Cuckoo search: recent advances and applications. Neural Comput Appl 24(1):169–174
39. Yang X-S (2010) Nature-inspired metaheuristic algorithms. Luniver Press, Frome
40. Dorigo M, Birattari M (2017) Ant colony optimization. In: Sammut C, Webb GI (eds) Encyclopedia of machine learning and data mining. Springer, Boston
41. Dorigo M, Di Caro G (1999) Ant colony optimisation: a new meta-heuristic. In: Evolutionary computation, 1999. CEC 99. Proceedings of the 1999 Congress on, vol 2. IEEE, pp 1470–1477
42. Darquennes D (2005) Implementation and applications of ant colony algorithms. Masters, Faculty of Computer Science, University of Namur, Belgium
43. Yang X-S, He X (2013) Bat algorithm: literature review and applications. Int J Bio-Inspired Comput 5(3):141–149
44. Yang XS, Gandomi AH (2012) Bat algorithm: a novel approach for global engineering optimisation. Eng Comput 29(5):464–483
45. Chittka L, Thomson JD, Waser NM (1999) Flower constancy, insect psychology, and plant evolution. Naturwissenschaften 86(8):361–377
46. Fister JI, Yang XS, Fister I, Brest J, Fister D (2013) A brief review of nature-inspired algorithms for optimisation. arXiv preprint arXiv:1307.4186
47. Ezugwu AE, Adewumi OA (2017) Discrete symbiotic organisms search algorithm for travelling salesman problem. Expert Syst Appl 30(87):70–78
48. Valle YD, Venayagamoorthy GK, Mohagheghi S, Hernandez JC, Harley RG (2008) Particle swarm optimisation: basic concepts, variants and applications in power systems. IEEE Trans Evol Comput 12(2):171–195
49. Cui Z (2009) Alignment particle swarm optimisation. In: 2009 8th IEEE international conference on cognitive informatics, pp 497–501
50. Kennedy J, Eberhart R (1995) Particle swarm optimisation. In: Neural networks, 1995. Proceedings, IEEE international conference on, vol 4, pp 1942–1948
51. Storn R, Price K (1997) Differential evolution—a simple and efficient heuristic for global optimisation over continuous spaces. J Glob Optim 11(4):341–359
52. Das S, Suganthan PN (2011) Differential evolution: a survey of the state-of-the-art. IEEE Trans Evol Comput 15(1):4–31
53. Goldberg D (1989) Genetic algorithms in optimisation, search and machine learning. Addison-Wesley, Reading
54. Tereshko V, Loengarov A (2005) Collective decision making in honey-bee foraging dynamics. Comput Inf Syst 9(3):1
55. Von Frisch K (2014) Bees: their vision, chemical senses, and language. Cornell University Press, Ithaca
56. Bozorg-Haddad O, Solgi M, Lo HA (2017) Meta-heuristic and evolutionary algorithms for engineering optimisation, vol 294. Wiley, New York
57. Fister I Jr, Fister D, Fister I (2013) A comprehensive review of cuckoo search: variants and hybrids. Int J Math Model Numer Optim 4(4):387–409
58. Qu C, He W (2015) A double mutation Cuckoo Search algorithm for solving systems of nonlinear equations. Int J Hybrid Inf Technol 8(12):433–448
59. Wu Y-C, Lee W-P, Chien C-W (2011) Modified the performance of differential evolution algorithm with dual evolution strategy. In: International conference on machine learning and computing, pp 57–63
60. Carr J (2014) An introduction to genetic algorithms. Sr Proj 1:40
61. Bai Q (2010) Analysis of particle swarm optimisation algorithm. Comput Inf Sci 3(1):180
62. Ezugwu AE (2019) Enhanced symbiotic organisms search algorithm for unrelated parallel machines manufacturing scheduling with setup times. Knowl-Based Syst. https://doi.org/10.1016/j.knosys.2019.02.005
63. Ali N, Othman MA, Husain MN, Misran MH (2014) A review of firefly algorithm. ARPN J Eng Appl Sci 9(10):1732–1736
64. Selvi V, Umarani DR (2010) Comparative analysis of ant colony and particle swarm optimisation techniques. Int J Comput Appl (0975–8887) 5(4):1–6
65. Abreu N, Ajmal M, Kokkinogenis Z, Bozorg B (2011) Ant colony optimisation, 26. https://paginas.fe.up.pt/~mac/ensino/docs/DS20102011/Presentations/PopulationalMetaheuristics/ACO_Nuno_Muhammad_Zafeiris_Behdad.pdf. Accessed 18 Feb 2018
66. Mohan N, Sivaraj R, Priya RD (2016) A comprehensive review of bat algorithm and its applications to various optimisation problems. Asian J Res Soc Sci Humanit 6(11):676–690
67. Xiao-hua S, Chun-ming Y (2013) Application of bat algorithm to permutation flow-shop scheduling problem. Ind Eng J 1:022
68. Yuce B, Packianather MS, Mastrocinque E, Pham DT, Lambiase A (2013) Honey bees inspired optimisation method: the bees algorithm. Insects 4(4):646–662
69. Balasubramani K, Marcus K (2014) A study on flower pollination algorithm and its applications. Int J Appl Innov Eng Manag 3(11):230–235
70. Wang Y, Li D, Lu Y, Cheng Z, Gao Y (2017) Improved flower pollination algorithm based on mutation strategy. In: Intelligent

human–machine systems and cybernetics (IHMSC), 2017 9th international conference on, 2017, pp 337–342
71. Yan G, Li C (2011) An effective refinement artificial bee colony optimisation algorithm based on chaotic search and application for PID control tuning. J Comput Inf Syst 7(9):3309–3316
72. Ahmadi M, Mojallali H (2012) Chaotic invasive weed optimisation algorithm with application to parameter estimation of chaotic systems. Chaos Solitons Fractals 45(9–10):1108–1120
73. Toksari MD (2016) A hybrid algorithm of Ant Colony Optimisation (ACO) and Iterated Local Search (ILS) for estimating electricity domestic consumption: case of Turkey. Int J Electr Power Energy Syst 78:776–782
74. Abrandao.com (2017) Genetic algorithms in PHP code. http://www.abrandao.com/2015/01/simple-php-genetic-algorithm/. Accessed 12 Dec 2017