Manifold Learning for Efficient Gravitational Search Algorithm

Chen Giladi[a,b], Avishai Sintov[c]

[a]*Department of Physics, Ben-Gurion University of the Negev, Beer-Sheva, Israel*
[b]*Department of Mechanical Engineering, Sami-Shamoon College of Engineering, Ashdod, Israel*
[c]*School of Mechanical Engineering, Tel-Aviv University, Tel-Aviv, Israel*

Abstract

Metaheuristic algorithms provide a practical tool for optimization in a high-dimensional search space. Some mimic phenomenons of nature such as swarms and flocks. Prominent one is the Gravitational Search Algorithm (GSA) inspired by Newton's law of gravity to manipulate agents modeled as point masses in the search space. The law of gravity states that interaction forces are inversely proportional to the squared distance in the Euclidean space between two objects. In this paper we claim that when the set of solutions lies in a lower-dimensional manifold, the Euclidean distance would yield unfitted forces and bias in the results, thus causing suboptimal and slower convergence. We propose to modify the algorithm and utilize geodesic distances gained through manifold learning via diffusion maps. In addition, we incorporate elitism by storing exploration data. We show the high performance of this approach in terms of the final solution value and the rate of convergence compared to other meta-heuristic algorithms including the original GSA. In this paper we also provide a comparative analysis of the state-of-the-art optimization algorithms on a large set of standard benchmark functions.

Keywords: Optimization, Metaheuristic algorithm, Gravitational Search Algorithm

1. Introduction

In many fields, there is an increasing need for solutions to high-dimensional real-life optimization problems. Optimization is required in many domains such as in finding the right design parameters for multi-objective power distribution feeder while considering several criteria [27], searching for high-energy particles [1], and dynamic locomotion [14]. One powerful branch of optimization algorithms that has considerably grown in the past two decades is known as *meta-heuristic* algorithms. These algorithms try to model physical or biological processes inspired by various tasks such as hunting, defence, navigation, foraging and lowering energy levels, which inherently solve high-dimensional optimization problems.

Previous research [6] has established that there are two main attributes to metaheuristic algorithms. The first aspect is stochastic behavior. Deterministic solvers find the same nearby local optimum for the same initial starting points. In contrast, metaheuristics incorporate randomness and thus, capable of avoiding getting trapped in local optima in the search for a global solution. However, in some particular cases they can still get trapped. Hence, their overall performance and applicability are reduced. Therefore, the second fundamental property for effectively solving high-dimensional optimization problems is the right balance between exploration and exploitation. Exploration enables the search for a global solution while acquiring more information. Exploitation uses current knowledge and performs local search for finding the optimum around good solutions [8].

Early metaheuristic algorithms include the Genetic Algorithm (GA) inspired by Darwins theory [15, 26], Simulated Annealing (SA) based on thermodynamic laws [19] and the Particle Swarm Optimization (PSO) inspired by animal flocks such as birds and fish [5, 18]. More recent algorithms are the Bacteria Foraging Optimization (BFO) mimicking the way bacteria search for nutrients [28], the Ant Lion Optimizer (ALO) which mimics the hunting mechanism of Antlions in nature [24]. Similarly, the Gravitational Search Algorithm (GSA) [30] and its different variations [10, 13] are inspired by the laws of gravity and motion. Additional metaheuristic algorithms can be reviewed in [11, 16, 17, 34, 25].

A more related metaheuristic algorithm could be reviewed in [23]. The algorithm, termed the Gray Wolf Optimizer (GWO), employs the idea of feature selection for exploring the data to eliminate irrelevant and redundant data while searching for the optimal solution. A classification accuracy-based fitness function was proposed, to explore regions of the complex search space. The author compares this algorithm with the PSO and GA over a set of benchmark machine learning data repository. Dimensionality reduction is also used in [22] to boost the performance of a Gaussian process model.

The standard GSA [30] uses the Euclidean space to calculate forces between agents based on Newton's law of gravity. However, in many problems, the set of solutions for the optimization problem lies in a lower dimensional subspace embedded in the ambient search space. Using a Euclidean metric may result in improper forces, may bias results, and trap or delay agents in local optima. Therefore and inspired by GSA, in this work we present a modified algorithm termed Curved Space Gravitational Search Algorithm (CSGSA). We propose working in the lower-dimensional subspace learned by an unsupervised machine learning algorithm. Through manifold learning we find the geodesic distances across the solution subspace which offer more fitted forces between agents.

Some semi-supervised algorithms also make similar assumptions, as in the our manifold learning approach, of a smooth manifold structure in the data. For example, Chapelle et al. [7] use this concept to find middle ground between having all training set labeled and no labels at all. Similarly, Belkin and Niyogi [3] assume that a high dimensional dataset used for a classification problem actually resided on a lower-dimensional manifold. The work suggests utilizing this data structure to overcome the tedious task of class labeling. However, there is no link between our work on meta-heuristic search algorithms to classification. Nevertheless, the fair assumption that real life high dimensional problems lie on a lower-dimnesional submanifold holds. The same idea can be reviewed in [31].

Along with the manifold learning, we add elitism to the algorithm by incorporating a simple memory-based approach. Consequently, this work provides an important opportunity to advance the understanding of combining two domains: metaheuristic algorithms and unsupervised learning. To support our contributions, we performed an extensive comparative study comparing our proposed algorithm to the state-of-the-art, GSA, ALO and PSO. The comparison is done through a large set of standard benchmarking functions, allowing effective analysis. The comparative analysis also provides an insight into the performance of prominent metaheuristic algorithms in various functions and may assist in future choices of algorithms. We note that we do not perform complexity comparison between the algorithms since complexity analysis for such meta-heuristic optimization algorithms simply do not exist due to their stochastic nature.

The paper is organized as follows. Section 2 introduces the original GSA algorithm and address cases in which the algorithm can bias the results. In Section 3, the CSGSA and its characteristics are described. Section 4 provides a brief overview of two state-of-the-art approaches to be used in the comparative study of Section 5.

2. Gravitational Search Algorithm (GSA)

A continuous parameter maximization problem for a given objective function of the form $f : \mathcal{X} \to \mathbb{R}$ with $\mathcal{X} \subseteq \mathbb{R}^n$, is defined as finding some global optimum $\mathbf{x}^* \in \mathcal{X}$ such that for any $\mathbf{x} \in \mathcal{X}$, $f(\mathbf{x}^*) \geq f(\mathbf{x})$ is satisfied. We note that without loss of generality, in the theoretic discussion we refer to a maximization problem. Nevertheless, a minimization problem can easily be addressed. Next, we present the original GSA algorithm as proposed in [30] followed by a discussion of cases in which the algorithm fails to perform well.

2.1. Algorithm

In this section we briefly present the GSA algorithm. Readers may review the complete algorithm in [30]. In GSA, candidate solutions are modelled as point mass objects, termed *agents*, in the high-dimensional search space. All agents attract each other based on the law of gravity, i.e., the force acting between two agents is inversely proportional to the product of their masses divided by the square of the distance between them.

The GSA algorithm is initialized with N random agents where the position $\mathbf{x}_i \in \mathcal{X}$ of each is given by

$$\mathbf{x}_i = \left(x_i^1, \ldots, x_i^n\right) \quad for \quad i = 1, 2, \ldots, N, \tag{1}$$

where x_i^d is the d^{th} vector component of the i-th agent. Each agent is allowed to move in an n-dimensional hyper-rectangle in $\mathcal{X}$ defined by the lower and upper bounds of the problem. At iteration t, the force exerted on agent j by agent i is

$$\mathbf{F}_{ij}(t) = G(t)\frac{M_i(t)M_j(t)}{R_{ij}(t)}\left(\mathbf{x}_j(t) - \mathbf{x}_i(t)\right) \tag{2}$$

where $M_i(t)$ and $G(t)$ are the mass of the i^{th} agent and the gravitational constant at time t, respectively. R_{ij} is the Euclidean distance between the two agents

$$R_{ij}(t) = \|\mathbf{x}_i(t) - \mathbf{x}_j(t)\|_2. \tag{3}$$

By lapse of iterations, the mass of an agent changes dynamically, and is calculated according to

$$M_i(t) = \frac{m_i(t)}{\sum_{j=1}^{N} m_j(t)} \tag{4}$$

where

$$m_i(t) = \frac{fit_i(t) - worst(t)}{best(t) - worst(t)}, \tag{5}$$

and $fit_i(t)$ is the fitness value or cost of agent i at time t, i.e., $fit_i(t) = f(\mathbf{x}_i(t))$. Thus, we collect the best and worst fitness values according to

$$best(t) = \max_{i \in i, \ldots N} fit_i(t) \quad \text{and} \quad worst(t) = \min_{i \in i, \ldots N} fit_i(t), \tag{6}$$

to determine the mass of each agent. In this way, masses with better fitness values will be heavier. Consequently, heavier agents hold better solutions and thus, are harder to be influenced while contributing better exploitation.

The algorithm must avoid trapping the agents in local optima. Thus, at first it is important to allow the agents to move more freely across the search space, i.e., enable more exploration. After

each iteration, the exploration should decrease while increasing exploitation. The way to enforce and control this behavior is by changing the gravitational constant over time $G(t)$. The update rule for this constant can be any decaying function. We choose an exponential decay in our work, which obeys the following

$$G(t) = G(t_o)exp\left(-\alpha\frac{t}{t_{max}}\right) \tag{7}$$

where $G(t_o)$ is the initial gravitational constant and α controls the decaying rate. Parameter t_{max} is user-defined and denotes the maximum number of iteration steps in the search.

The total force acting on mass i in the d^{th} dimension at time t is given by

$$F_i^d(t) = \sum_{j\in KBest, j\neq i}^{N} rand_j F_{ij}^d(t), \tag{8}$$

where $rand_j$ is a random number in the interval $[0, 1]$ and $KBest$ is the group of K agents with the best fitness values. As the exploration decays in favor of exploitation, the number of agents in $Kbest$ decreases such that only they attract others. The acceleration of particle i in the d^{th} dimension is calculated by applying Newton's second law

$$a_i^d(t) = \frac{F_i^d(t)}{M_i(t)}. \tag{9}$$

In order to insert a stochastic characteristic to the algorithm, the velocity update of an agent is the addition of the current agent's velocity to the imposed acceleration multiplied by a random variable in the interval $[0, 1]$ as follows

$$v_i^d(t+1) = rand_i\left(v_i^d(t) + a_i^d(t)\right). \tag{10}$$

We note that a constant of one is multiplied before the acceleration term to match the units. Finally, the new update location of particle i in the d^{th} dimension is

$$x_i^d(t+1) = x_i^d(t) + v_i^d(t+1). \tag{11}$$

The exploration involves not only the randomization of the initial population but also the acting forces as can be seen in (8). Consequently, the agents would explore the search space according to the overall attraction forces of this multi-body problem. An interesting part is the movement dynamics of the heavier agent. The heavier an agent, the less it would move. This would inherently correspond to good exploitation ability leading to a better solution in the search space.

2.2. Problem

In many real-life problems, the dimension of $\mathcal{X}$ is high and some dimensions have greater effect on the optimality search. Further, it is not a priori known which dimensions offer the fastest and better global solution. Moreover, the problem becomes more severe as the number of optimization parameters increases. This is a general problem recognized as one of the main challenges faced by high-dimensional optimization algorithms and is known as the curse of dimensionality [4].

The curse of dimensionality has additional implications in GSA when some dimensions are correlated. The transfer of information between agents is based on the law of gravity. Within this framework, there are two possibilities of transferring information. The first is the product of

masses in the numerator of (2). A better fitness value results in a heavier mass with stronger ability to pull other agents. This will result in higher acceleration toward the fitter solution. The distance in the denominator of (2) is the second mean by which information is transferred between agents. The closer two agents are, the stronger the attraction forces between them. In this paper, we argue that these forces are fit only if the dimensions are uncorrelated. If there are correlations such that the solutions lie in a lower-dimensional manifold in the search space, then the sampling used in this stochastic optimization algorithm might end up with very biased results. Moreover, it may occur that the data lies on some manifold such that GSA wrongly invokes excessive forces between two agents. The two agents may appear close to each other whereas on the actual manifold, they are far away.

Figure 1 illustrates two planar examples where the resulting forces interfere with the maximization. In both examples, agents M_1 and M_2 are close in terms of the Euclidean distance in $\mathcal{X}$ and thus, attract each other with a relatively large force. In Figure 1a, M_2 prevents M_1 from climbing to the maximum point. In Figure 1b, the information transfer from the fittest agent M_3 is overtaken by the attraction force F_{12} which is quite large due to the short Euclidean distance between M_1 and M_2. Similarly, in the higher-dimensional Swiss-roll example of Figure 2, the distances between the agents are calculated according to the Euclidean distance and result in unfitted attraction forces. However, calculating the distances across the Swiss-roll manifold may provide more suited forces. In the following section we discuss this approach and the Curved Space Gravitational Search Algorithm (CSGSA).

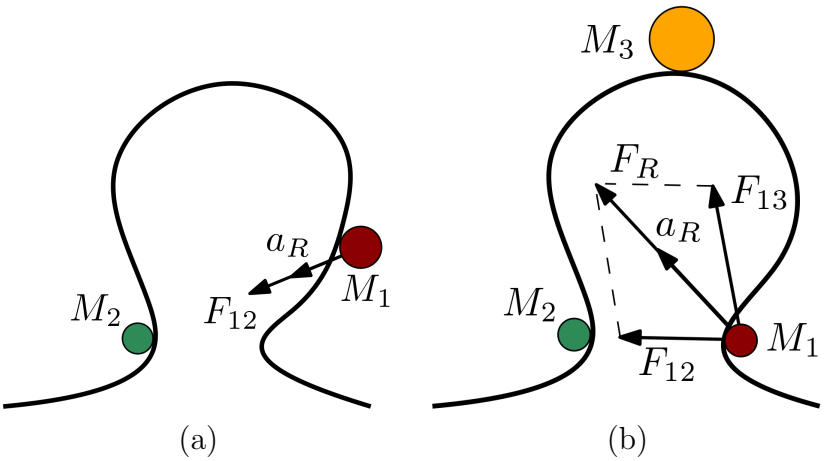


Figure 1: Two examples of maximization problems where agents M_1 and M_2 prevent one another from climbing up and advance toward the maximum.

3. Curved Space Gravitational Search Algorithm

In this Section we introduce our modifications for the GSA to address the issues discussed in the previous section.

3.1. Approach

We have argued that considering Euclidean distances in the calculation of the attraction forces would result in unfitted forces and bias in the results. One method that has evolved to help solve such problems is based on nonlinear dimensionality reduction of data in high-dimensional spaces.

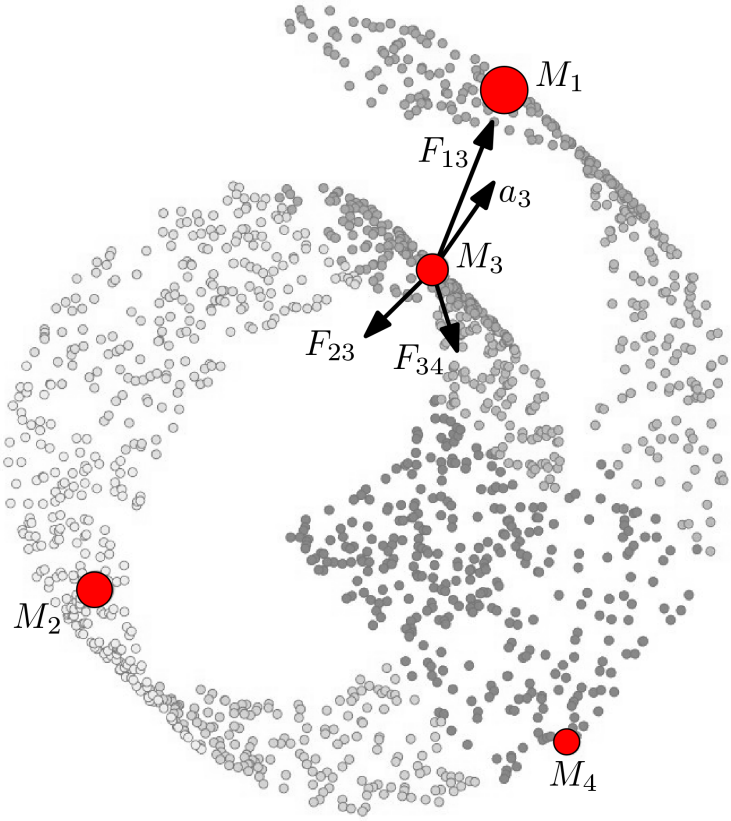


Figure 2: An example of the solution subspace in the form of a Swiss-roll where agent M_1 has too much influence on M_3 due to attraction forces computed according to the Euclidean distance in $\mathcal{X}$. Agents may be influenced by forces that do not take the real distances across the manifold, thus causing biased resultant forces.

The basic idea in nonlinear dimensionality reduction is to find a low-dimensional manifold embedded in a high-dimensional space. One interesting approach uses a combination of a metaheuristic algorithm and an unsupervised learning technique, specifically, manifold learning [2, 3, 33]. The idea is to map samples from a higher dimensional space to a lower subspace and to perform the search in the latter.

As was pointed out previously, the agents are assumed to live on a manifold embedded in high-dimensional search space. When calculating the distance between two agents, one does not need to obey the standard Newton's law of gravity. Specifically, the Euclidean distance is not always suitable for the calculation of the attraction forces. Therefore, we focus on calculating the actual distance along the manifold where agents solely move on, i.e., geodesic distance. We propose to add an important step to the GSA. By utilizing diffusion maps [9], a known dimensionality reduction technique, we can uncover the subspace in which the population lives. We argue that this would serve as a better way to calculate the distance between two agents and therefore, provide a better estimation of the actual gravitational force exerted on each agent. Consequently, an improved update rule would be acquired more suited to the given problem. In essence, the key advancement proposed in this work is to dynamically modify the update rule of each agent and to transfer information between them based on their true distances across the manifold. The complete CSGSA is presented in Algorithm 1.

In addition to manifold learning, we propose to improve exploration by randomly generating N

new agents in each iteration. However, new sampling may result in the lost of the original samples distribution, hence, increasing the chances of repeated calculations. Thus, in opposed to GSA, we propose to incorporate an element of elitism. We create a candidate container H that stores the last status of all sampled agents. In each iteration, N new agents are sampled, evaluated and added to H. Then, the fittest N agents are extracted from the database and used for exploration and exploitation. This step takes previous exploration history to consideration and ensures the update of the most relevant candidate solutions. Only the most fitted agents are allowed to evolve.

Algorithm 1: Curved space gravitational search algorithm

```
while criterion is not satisfied do
    Randomly initialize N new agents.;
    Calculate fitness fit_i for each agent i = 1, ..., N.;
    Update new agents to database H.;
    Choose N best agents from H.;
    Calculate best_i, worst_i and M_i for each agent i = 1, ..., N.;
    Update gravitational constant G.;
    Learn manifold and calculate relative distances.;
    Calculate acceleration in gravitational field for each agent.;
    Update velocities and positions.;
return best solution;
```

3.2. Dimensionality Reduction by Diffusion Maps

The learning of the manifold could be done via various proposed algorithms for dimensionality reduction. In this paper, we choose to use *Diffusion Maps* as a tool to learn the distance along this manifold as presented by Coifman and Lafon [9]. Diffusion maps are a graph-based dimensionality reduction method with applications such as image processing [12] and data clustering [20]. A diffusion map is an embedding in the Euclidean space $\mathbb{R}^n$ and thus, the diffusion distance inherits all the metric properties of $\mathbb{R}^n$. Therefore, the diffusion map is an efficient method for acquiring a metric subspace corresponding to the non-linear manifold in the Euclidean space. The general idea is to find the underlying manifold that the data has been sampled from. Coifman and Lafon found that through the computation of the eigenvectors and eigenvalues of a diffusion operator on the data, one could estimate the diffusion distance between probability distributions centered at two points in the high-dimensional space.

Suppose we have N agents. For approximating the probability of transition from one agent's position to the next, we define the kernel function

$$k(\mathbf{x}_i, \mathbf{x}_j) = \exp\left(\frac{-\|\mathbf{x}_i - \mathbf{x}_j\|^2}{\gamma}\right) \tag{12}$$

to be Gaussian with a decaying rate of γ. This kernel establishes prior local geometry information of the agents. In order to have a probability function of taking a step from the i-th agent to the j-th agent, we calculate the reversible Markov chain on the dataset as follows

$$\text{connectivity}(\mathbf{x}_i, \mathbf{x}_j) = p(\mathbf{x}_i, \mathbf{x}_j) = \frac{k(\mathbf{x}_i, \mathbf{x}_j)}{\sum_i k(\mathbf{x}_i, \mathbf{x}_j)}. \tag{13}$$

We now define the normalized diffusion matrix P such that $P_{i,j} = p(\mathbf{x}_i, \mathbf{x}_j)$. Each component $P_{i,j}$ encapsulates the local knowledge of the connectivity between $\mathbf{x}_i$ and $\mathbf{x}_j$.

Dimensionality reduction is done by neglecting certain dimensions in the diffusion space of the normalized diffusion matrix P. In practice, we find the eigenvectors of P by solving

$$P\mathbf{z} = \lambda\mathbf{z} \tag{14}$$

where $\mathbf{z} \in \mathbb{R}^N$ and $\lambda \in \mathbb{R}$. Thus, $\mathbf{z}_i$, $i = 1, ..., N$ are the eigenvectors of P corresponding to the eigenvalues $1 \geq \lambda_1 \geq \ldots \geq \lambda_N$. We choose only the $m < n$ dimensions associated with the dominant eigenvectors of P and map $\mathbf{x}_i \in \mathcal{X}$ to $\mathbf{y}_i \in \mathbb{R}^m$ according to

$$\mathbf{y}_i = (z_1^i, \ldots, z_m^i)^T, \quad i = 1, \ldots, N, \tag{15}$$

where z_j^i is the i-th element of $\mathbf{z}_j$. The basic diffusion-mapping algorithm is described in Algorithm 2 [29].

Algorithm 2: Diffusion map algorithm

1 **Input:** High dimensional data set $\mathbf{x_i}$, $i = 1, \ldots, N$.;
2 Define kernel $k(x, y)$.;
3 Create a diffusion matrix P.;
4 Calculate the eigenvectors of the diffusion matrix P.;
5 Map to the m-dimensional diffusion space using the m dominant eigen-vectors according to (15).;
6 **Return** Lower dimensional data set $\mathbf{y}_i$, $i = 1, \ldots, N$.;

In general, the higher the dimensionality, the more difficult it becomes to sample the space. In order to correctly map the high-dimensional data set to a lower-dimensional subspace, a nonlinear dimensionality reduction technique requires as much data as possible. Many researchers have utilized metaheuristic algorithms in a sequential manner. At each iteration of the algorithm, perform three main steps: self-adaptation, cooperation and competition between different agents [21]. The last step is highly important. As in nature, low genetic diversity increases the chances of future generations of offspring to have the same attributes as their parents. Thus, this results in reduced robustness to environmental changes [21]. In other words, nature's capability to adapt to changes in the environment improves with the population growth. Metaheuristic algorithms, which are also population-based, improve according to the number of agents. Although the size of the population needed to efficiently solve a specific problem may vary from one problem to another, a general rule of thumb is, the larger the population the better.

4. Metaheuristic state-of-the-art algorithms

In the comparative study of Section 5, we compare the CSGSA to the Particle Swarm Optimization (PSO), to the Ant Lion Optimization (ALO) and to the original GSA. PSO and ALO are briefly reviewed in this section prior to the comparative analysis.

4.1. Particle Swarm Optimization

Particle Swarm Optimization (PSO) [18] is inspired by bird flocking and fish schooling in nature. A flock of birds or a particle swarm operate according to the fitness information obtained

from the environment. In this way, particles update their position according to their individual best-known fitness and the overall groups best solution. In PSO, the position and velocity of each individual are calculated and updated as follows:

$$x_i^d(t+1) = x_i^d(t) + v_i^d(t+1) \tag{16}$$

$$v_i^d(t+1) = w(t)v_i^d(t) + c_1 r_{i1}(pbest_i^d - x_i^d(t)) + c_2 r_{i2}(gbest^d - x_i^d(t)) \tag{17}$$

where r_{i1} and r_{i2} are two random variable in the interval $[0, 1]$, c_1 and c_2 are two positive constants, and w is the inertia parameter. The position and velocity of the i-th particle are represented by $\mathbf{x}_i = (x_i^1, \ldots, x_i^n)$ and $\mathbf{v}_i = (v_i^1, \ldots, v_i^n)$, respectively. Also, the best solution of the i-th particle is stored as $pbest_i = (pbest_i^1, \ldots, pbest_i^n)$ and the best result among all of the population is stored as $gbest = (gbest^1, \ldots, gbest^n)$. PSO resembles GSA with the use of agents in the solution space. However, PSO incorporates elitism and relies on previous fitness information whereas GSA only uses the current status of the agents.

4.2. Ant Lion Optimization (ALO)

The Ant Lion (Antlion) [24] optimizer is inspired by the interactive dynamics of antlions and ants in nature. As such, ants are allowed to move freely over the search space and in a stochastic manner. The position of a single ant is equivalent to a solution of an optimization problem. Ants move in the solution space with random walks and their position is calculated as follows

$$x_i^d = [0, cumsum(2r(t_1) - 1), \ldots, cumsum(2r(t_k) - 1)] \tag{18}$$

where *cumsum* calculates the cumulative sum, k is the maximum number of iterations, t_j is the j-th time step of the random walk. $r(t)$ is a stochastic function defined as follows:

$$r(t) = \begin{cases} 1, & rand() \leq 0.5 \\ 0, & otherwise \end{cases} \tag{19}$$

where t is the time step and $rand()$ is a random number uniformly generated in the interval of $[0, 1]$.

Antlions spread in the search space lay pits to trap ants where each pit size is proportional to the fitness of the antlion. Thus, larger pits have a higher probability to trap ants. Ants are required to move within a hyper-sphere around a selected antlion. When an antlion tries to catch an ant, it throws sand to the outer edge of the pit to create a small avalanche of sand and to make it harder for the ant to escape. This behavior is modeled by adaptively decreasing the ant's random walk hyper-sphere. Consequently, the ant is forced to slide inside the pit and thus, exploitation in the vicinity of the antlion is enabled. If an ant becomes fitter than its nearest antlion, the antlion reposition itself to the position of the ant by

$$Antlion_j^t = Ant_i^t, \text{ if } f(Ant_j^t) > f(Antlion_j^t) \tag{20}$$

where $Antlion_j^t$ and Ant_i^t are the j-th antlion and i-th ant's position, respectively, at time t, and $f(\cdot)$ is the fitness function. This process mimics the antlion catching its prey. Through iterations, the best antlion is saved (position and fitness) and updated until termination criterion is satisfied. The best antlion at the end of the run is considered as the global solution.

5. Experimental results

In this section we evaluate the performance of the proposed algorithm and provide a comparative analysis. The CSGSA is compared to ALO, PSO and the original GSA. In our analysis, we pay attention to two important attributes of the performance: the final fitness value and the rate of convergence. For that matter, the algorithms are benchmarked on a set of standard functions presented in Section 5.1 followed by a comparative study in Section 5.2.

5.1. Benchmark functions

The comparative analysis was performed by benchmarking 47 standard test functions taken from [32]. The functions are divided into six groups: many local minima (Table 1), bowl-shaped (Table 2), plate-shaped (Table 3), valley-shaped (Table 4), steep ridges or drops (Table 5) and various other functions (Table 6). Tables 1-6 present the functions, the global minimum value of each function and the search subset in $\mathcal{X}$. The standard names of the functions and additional information are given in Tables A.1 and A.2 of Appendix A, respectively. Plots of the functions are seen in Figures A.1-A.6. Some of the functions have a multitude of local optima. Thus, the algorithms are tested in their ability to balance between exploration and exploitation to find the global optimum.

5.2. Results

We compared the proposed CSGSA with the PSO, ALO and GSA algorithms using the mentioned set of benchmark functions. For fair and easy comparison, we use the same hyper-parameters used in the relavent literature, including the number of agents. Therefore, in all cases, population size is chosen to be $N = 50$. In the multi-dimensional benchmark functions, the dimension is set to $d = 30$ and the maximum number of iterations is 1,000. The results are averaged over 30 trials. In PSO, the constants used are $c_1 = 1.5$, $c_2 = 2$ and $w = 1$. All algorithms were implemented and tested in Matlab.

The Wilcoxon test is a non-parametric statistical way of computing the sampling distribution for any test statistic. It tests the null hypothesis as to whether two sets of solutions are significantly different. The Wilcoxon test outputs a parameter termed p-value that determines the significance of the results for whether the null hypothesis is true. In this work, the test compares the performance of two algorithms applied to the same set of benchmark functions. Thus, the difference between two algorithms is statistically significant if the p-value is less than 0.05. All algorithms are compared to the CSGSA.

The average best-so-far, standard deviation, p-value and average number of iterations to convergence for each algorithm and benchmark function are presented in Tables 7-12. The average number of iterations leading to convergence is calculated as the number of iterations in which the relative error is 2%, that is, the following condition is satisfied

$$\left| \frac{best(t) - best(t_{max})}{best(t_{max})} \right| < 0.02, \; best(t) = \max_{i \in 1,\dots N} f(\mathbf{x}_i(t)). \tag{21}$$

To summarize the results, Table 13 presents the performance of CSGSA compared to the other algorithms with regards to the group of functions. It shows the percentage in which CSGSA performs equally or better in terms of the converged fitness value and the required number of iterations to convergence. Since each of the metaheuristic algorithms is stochastic, there are usually different final values after N iterations. Here also and for determining if two sets of results

Table 1: Many Local Minima Functions

Function	Global Minimum	Search Space
$F_1(x) = -a\exp(-b\sqrt{\frac{1}{d}\sum_{i=1}^{d} x_i^2}) - \exp(\frac{1}{d}\sum_{i=1}^{d}\cos(cx_i)) + a + \exp(1)$	$F(x^*) = 0,$ $at\ x^* = (0, ..., 0)$	$x_i \in [-32.768, 32.768],$ $for\ all\ i = 1, ..., d$
$F_2(x) = 100\sqrt{\lvert x_2 - 0.01x_1^2\rvert} + 0.01\lvert x_1 + 10\rvert$	$F(x^*) = 0,$ $at\ x^* = (-10, 1)$	$x_1 \in [-15, -5],$ $x_2 \in [-3, 3]$
$F_3(x) = -0.0001\left(\left\lvert\sin(x_1)\sin(x_2)\exp\left(\left\lvert 100 - \frac{\sqrt{x_1^2+x_2^2}}{\pi}\right\rvert\right)\right\rvert + 1\right)^{0.1}$	$F(x^*) = -2.06261,$ $at\ x^* = (1.3491, -1.3491),$ $(1.3491, 1.3491),$ $(-1.3491, 1.3491)\ and$ $(-1.3491, -1.3491)$	$x_i \in [-10, 10],$ $for\ all\ i = 1, 2$
$F_4(x) = -\frac{1+\cos(12\sqrt{x_1^2+x_2^2})}{0.5(x_1^2+x_2^2)+2}$	$F(x^*) = -1,$ $at\ x^* = (0, 0)$	$x_i \in [-5.12, 5.12],$ $for\ all\ i = 1, 2$
$F_5(x) = -(x_2 + 47)\sin\left(\sqrt{\lvert x_2 + \frac{x_1}{2} + 47\rvert}\right) - x_1\sin(\sqrt{\lvert x_1 - (x_2 + 47)\rvert})$	$F(x^*) = -959.6407,$ $at\ x^* = (512, 404.2319)$	$x_i \in [-512, 512],$ $for\ all\ i = 1, 2$
$F_6(x) = \frac{\sin(10\pi x)}{2x} + (x-1)^4$		$x \in [0.5, 2.5]$
$F_7(x) = \sum_{i=1}^{d}\frac{x_i^2}{4000} - \prod_{i=1}^{d}\cos\left(\frac{x_i}{\sqrt{i}}\right) + 1$	$F(x^*) = 0,$ $at\ x^* = (0, ..., 0)$	$x_i \in [-600, 600],$ $for\ all\ i = 1, ..., d$
$F_8(x) = -\left\lvert\sin(x_1)\cos(x_2)\exp\left(\left\lvert 1 - \frac{\sqrt{x_1^2+x_2^2}}{\pi}\right\rvert\right)\right\rvert$	$F(x^*) = -19.2085,$ $at\ x^* = (8.05502, 9.66459),$ $(8.05502, -9.66459),$ $(-8.05502, 9.66459)\ and$ $(-8.05502, -9.66459)$	$x_i \in [-10, 10],$ $for\ all\ i = 1, 2$
$F_9(x) = \sum_{i=1}^{m} c_i\exp\left(-\frac{1}{\pi}\sum_{j=1}^{d}(x_j - A_{ij})^2\right)\cos\left(\pi\sum_{j=1}^{d}(x_j - A_{ij})^2\right)$		$x_i \in [0, 10],$ $for\ all\ i = 1, ..., d$
$F_{10}(x) = \sin^2(\pi w_1) + \sum_{i=1}^{d-1}(w_i - 1)^2[1 + 10\sin^2(\pi w_i + 1)] + (w_d - 1)^2$ $[1 + \sin^2(2\pi w_d)]$	$F(x^*) = 0,$ $at\ x^* = (1, ..., 1)$	$x_i \in [-10, 10],$ $for\ all\ i = 1, ..., d$
$F_{11}(x) = \sin^2(3\pi x_1) + (x_1 - 1)^2[1 + \sin^2(3\pi x_2)] + (x_2 - 1)^2[1 + \sin^2(2\pi x_2)]$	$F(x^*) = 0,$ $at\ x^* = (1, 1)$	$x_i \in [-10, 10],$ $for\ all\ i = 1, 2$
$F_{12}(x) = 10d + \sum_{i=1}^{d}[x_i^2 - 10\cos(2\pi x_i)]$	$F(x^*) = 0,$ $at\ x^* = (0, ..., 0)$	$x_i \in [-5.12, 5.12],$ $for\ all\ i = 1, ..., d$
$F_{13}(x) = 0.5 + \frac{\sin^2(x_1^2 - x_2^2) - 0.5}{[1+0.001(x_1^2 - x_2^2)]^2}$	$F(x^*) = 0,$ $at\ x^* = (0, 0)$	$x_i \in [-100, 100],$ $for\ all\ i = 1, 2$
$F_{14}(x) = 0.5 + \frac{\cos(\sin(\lvert x_1^2 - x_2^2\rvert)) - 0.5}{[1+0.001(x_1^2 - x_2^2)]^2}$		$x_i \in [-100, 100],$ $for\ all\ i = 1, 2$
$F_{15}(x) = 418.9829d - \sum_{i=1}^{d} x_i\sin(\sqrt{\lvert x_i\rvert})$	$F(x^*) = 0, at$ $x^* = (420.9687, ..., 420.9687)$	$x_i \in [-500, 500],$ $for\ all\ i = 1, ..., d$
$F_{16}(x) = \left(\sum_{i=1}^{5} i\cos((i+1)x_1 + i)\right)\left(\sum_{i=1}^{5} i\cos((i+1)x_2 + i)\right)$	$F(x^*) = -186.7309$	$x_i \in [-10, 10],$ $for\ all\ i = 1, 2$

Table 2: Bowl-Shaped Functions

Function	Global Minimum	Search Space
$F_{17_1}(x) = x_1^2 + 2x_2^2 - 0.3\cos(3\pi x_1) - 0.4\cos(4\pi x_2) + 0.7,$	$F_j(x^*) = 0, at\ x^* = (0,0),$	$x_i \in [-100, 100],$
$F_{17_2}(x) = x_1^2 + 2x_2^2 - 0.3\cos(3\pi x_1)\cos(4\pi x_2) + 0.3,$	$for\ all\ j = 1,2,3$	$for\ all\ i = 1,2$
$F_{17_3}(x) = x_1^2 + 2x_2^2 - 0.3\cos(3\pi x_1 + 4\pi x_2) + 0.3$		
$F_{18}(x) = \sum_{i=1}^{d}\left(\sum_{j=1}^{d}(j+\beta)\left(x_j^i - \frac{1}{j^i}\right)\right)^2$	$F(x^*) = 0,$	$x_i \in [-d, d],$
	$at\ x^* = (1, \frac{1}{2}, ...\frac{1}{d})$	$for\ all\ i = 1, ..., d$
$F_{19}(x) = \sum_{i=1}^{d}\sum_{j=1}^{i} x_j^2$	$F(x^*) = 0,$	$x_i \in [-65.536, 65.536],$
	$at\ x^* = (0, ..., 0)$	$for\ all\ i = 1, ..., d$
$F_{20}(x) = \sum_{i=1}^{d} x_i^2$	$F(x^*) = 0,$	$x_i \in [-5.12, 5.12],$
	$at\ x^* = (0, ..., 0)$	$for\ all\ i = 1, ..., d$
$F_{21}(x) = \sum_{i=1}^{d} \lvert x_i\rvert^{i+1}$	$F(x^*) = 0,$	$x_i \in [-1, 1],$
	$at\ x^* = (0, ..., 0)$	$for\ all\ i = 1, ..., d$
$F_{22}(x) = \sum_{i=1}^{d} ix_i^2$	$F(x^*) = 0,$	$x_i \in [-10, 10],$
	$at\ x^* = (0, ..., 0)$	$for\ all\ i = 1, ..., d$
$F_{23}(x) = \sum_{i=1}^{d}(x_i - 1)^2 - \sum_{i=2}^{d} x_i x_{i-1}$	$F(x^*) = -d(d+4)(d-1)/6,$	$x_i \in [-d^2, d^2],$
	$at\ x_i = i(d+1-i),$	$for\ all\ i = 1, ..., d$
	$for\ all\ i = 1, 2, ..., d$	

Table 3: Plate-Shaped Functions

Function	Global Minimum	Search Space
$F_{24}(x) = (x_1 + 2x_2 - 7)^2 + (2x_1 + x_2 - 5)^2$	$F(x^*) = 0,$	$x_i \in [-10, 10],$
	$at\ x^* = (1, 3)$	$for\ all\ i = 1, 2$
$F_{25}(x) = 0.26(x_1^2 + x_2^2) - 0.48x_1x_2$	$F(x^*) = 0,$	$x_i \in [-10, 10],$
	$at\ x^* = (0, 0)$	$for\ all\ i = 1, 2$
$F_{26}(x) = \sin(x_1 + x_2) + (x_1 - x_2)^2 - 1.5x_1 + 2.5x_2 + 1$	$F(x^*) = -1.9133,$	$x_1 \in [-1.5, 4],$
	$at\ x^* = (-0.54719, -1.54719)$	$x_2 \in [-3, 4]$
$F_{27}(x) = \sum_{i=1}^{d}\left[\left(\sum_{j=1}^{d} x_j^i\right) - b_i\right]^2$		$x_i \in [0, d],$
		$for\ all\ i = 1, ..., d$
$F_{28}(x) = \sum_{i=1}^{d} x_i^2 + \left(\sum_{i=1}^{d} 0.5ix_i\right)^2 + \left(\sum_{i=1}^{d} 0.5ix_i\right)^4$	$F(x^*) = 0,$	$x_i \in [-5, 10],$
	$at\ x^* = (0, ..., 0)$	$for\ all\ i = 1, ..., d$

Table 4: Valley-Shaped Functions

Function	Global Minimum	Search Space
$F_{29}(x) = 2x_1^2 - 1.05x_1^4 + \frac{x_1^6}{6} + x_1x_2 + x_2^2$	$F(x^*) = 0,$	$x_i \in [-5, 5],$
	$at\ x^* = (0, 0)$	$for\ all\ i = 1, 2$
$F_{30}(x) = \left(4 - 2.1x_1^2 + \frac{x_1^4}{3}\right)x_1^2 + x_1x_2 + (-4 + 4x_2^2)x_2^2$	$F(x^*) = -1.0316, at$	$x_1 \in [-3, 3],\ x_2 \in [-2, 2]$
	$x^* = (0.0898, -0.7126)\ and$	
	$(-0.0898, 0.7126)$	
$F_{31}(x) = (x_1 - 1)^2 + \sum_{i=2}^{d} i(2x_i^2 - x_{i-1})^2$	$F(x^*) = 0,\ at\ x_i = 2^{-\frac{2^i-2}{2^i}},$	$x_i \in [-10, 10],$
	$for\ i = 1, \ldots, d$	$for\ all\ i = 1, ..., d$
$F_{32}(x) = \sum_{i=1}^{d-1}[100(x_{i+1} - x_i^2)^2 + (x_i - 1)^2]$	$F(x^*) = 0,$	$x_i \in [-5, 10],$
	$at\ x^* = (1, ..., 1)$	$for\ all\ i = 1, ..., d$

Table 5: Steep Ridges/Drops Functions

Function	Global Minimum	Search Space
$F_{33}(x) = \left(0.002 + \sum_{i=1}^{25} \frac{1}{i+(x_1-a_{1i})^6+(x_2-a_{2i})^6}\right)^{-1}$		$x_i \in [-65.536, 65.536]$, $for\ all\ i = 1, 2$
$F_{34}(x) = -\cos(x_1)\cos(x_2)\exp(-(x_1-\pi)^2-(x_2-\pi)^2)$	$F(x^*) = 0$, $at\ x^* = (\pi, \pi)$	$x_i \in [-100, 100]$, $for\ all\ i = 1, 2$
$F_{35}(x) = -\sum_{i=1}^{d} \sin(x_i)\sin^{2m}\left(\frac{ix_i^2}{\pi}\right)$	$at\ d = 2:\ F(x^*) = -1.8013$, $at\ x^* = (2.20, 1.57)$, $at\ d = 5:\ F(x^*) = -4.687658$, $at\ d = 10:\ F(x^*) = -9.66015$	$x_i \in [0, \pi]$, $for\ all\ i = 1, ..., d$

Table 6: Other Functions

Function	Global Minimum	Search Space
$F_{36}(x) = (1.5 - x_1 + x_1x_2)^2 + (2.25 - x_1 + x_1x_2^2)^2 + (2.625 - x_1 + x_1x_2^3)^2$	$F(x^*) = 0$, $at\ x^* = (3, 0.5)$	$x_i \in [-4.5, 4.5]$, $for\ all\ i = 1, 2$
$F_{37}(x) = a(x_2 - bx_1^2 + cx_1 - r)^2 + s(1-t)\cos(x_1) + s$	$F(x^*) = 0.397887$, $at\ x^* = (-\pi, 12.275)$, $(\pi, 12.275)\ and$ $(9.42478, 2.475)$	$x_1 \in [-5, 10]$, $x_2 \in [0, 15]$
$F_{38}(x) = 100(x_1^2 - x_2)^2 + (x_1-1)^2 + (x_3-1)^2 + 90(x_3^2 - x_4)^2 +$ $10.1((x_2-1)^2 + (x_4-1)^2) + 19.8(x_2-1)(x_4-1)$	$F(x^*) = 0$, $at\ x^* = (1, 1, 1, 1)$	$x_i \in [-10, 10]$, $for\ all\ i = 1, 2, 3, 4$
$F_{39}(x) = (6x-2)^2\sin(12x-4)$		$x \in [0, 1]$
$F_{40}(x) = [1 + (x_1+x_2+1)^2(19 - 14x_1 + 3x_1^2 - 14x_2 + 6x_1x_2 + 3x_2^2)]\times$ $[30 + (2x_1 - 3x_2)^2(18 - 32x_1 + 12x_1^2 + 48x_2 - 36x_1x_2 + 27x_2^2)]$	$F(x^*) = 3$, $at\ x^* = (0, -1)$	$x_i \in [-2, 2]$, $for\ all\ i = 1, 2$
$F_{41}(x) = -\sum_{i=1}^{4} \alpha_i \exp\left(-\sum_{j=1}^{3} A_{ij}(x_j - P_{ij})^2\right)$	$F(x^*) = -3.86278$, at $x^* = (0.114614, 0.555649, 0.852547)$	$x_i \in (0, 1)$, $for\ all\ i = 1, 2, 3$
$F_{42}(x) = \frac{1}{0.839}\left[1.1 - \sum_{i=1}^{4} \alpha_i \exp\left(-\sum_{j=1}^{4} A_{ij}(x_j - P_{ij})^2\right)\right]$		$x_i \in [0, 1]$, $for\ all\ i = 1, 2, 3, 4$
$F_{43}(x) = -\sum_{i=1}^{4} \alpha_i \exp\left(-\sum_{j=1}^{6} A_{ij}(x_j - P_{ij})^2\right)$	$F(x^*) = -3.32237$, at $x^* = (0.20169, 0.150011, 0.476874,$ $0.275332, 0.311652, 0.6573)$	$x_i \in (0, 1)$, $for\ all\ i = 1, \ldots, 6$
$F_{44}(x) = \sum_{i=1}^{d}\left(\sum_{j=1}^{d}(j^i + \beta)\left(\left(\frac{x_j}{j}\right)^i - 1\right)\right)^2$	$F(x^*) = 0$, $at\ x^* = (1, 2, ..., d)$	$x_i \in [-d, d]$, $for\ all\ i = 1, ..., d$
$F_{45}(x) = \sum_{i=1}^{d/4}[(x_{4i-3} + 10x_{4i-2})^2 + 5(x_{4i-1} - x_{4i})^2 +$ $(x_{4i-2} - 2x_{4i-1})^4 + 10(x_{4i-3} - x_{4i})^4]$	$F(x^*) = 0$, $at\ x^* = (0, ..., 0)$	$x_i \in [-4, 5]$, $for\ all\ i = 1, ..., d$
$F_{46}(x) = -\sum_{i=1}^{m}\left(\sum_{j=1}^{4}(x_j - C_{ji})^2 + \beta_i\right)^{-1}$	$at\ m = 5:\ F(x^*) = -10.1532$, $at\ x^* = (4, 4, 4, 4)$, $at\ m = 7:\ F(x^*) = -10.4029$, $at\ x^* = (4, 4, 4, 4)$, $at\ m = 10:\ F(x^*) = -10.5364$, $at\ x^* = (4, 4, 4, 4)$	$x_i \in [0, 10]$, $for\ all\ i = 1, 2, 3, 4$
$F_{47}(x) = \frac{1}{2}\sum_{i=1}^{d}(x_i^4 - 16x_i^2 + 5x_i)$	$F(x^*) = -39.16599d$, $at\ x^* = (-2.903534, \ldots, -2.903534)$	$x_i \in [-5, 5]$, $for\ all\ i = 1, ..., d$

are statistically significant, we used the Wilcoxon rank-sum test. The test was used both for the converged fitness value and for the rate for convergence.

When observing the converged fitness value and when comparing to ALO and GSA, in most cases CSGSA reaches the same optimum or better. Compared to PSO, CSGSA outperforms in some cases but not so distinctively. Nonetheless, it performs quite well compared to PSO and significantly outperforms ALO and GSA in the first group of functions with many local minima (1-16). This is an evident that considering geodesic distances provides a strong advantage. This strategy is more pronounced compared to using the Euclidean distance in cases of many local minima and step ridges. Figure 1a illustrates a clear example of biased results when sharing information between agents while considering the Euclidean distance. This type of property occurs in the mentioned functions and may lead to inferior behavior of the population. Thus, it is difficult for the agents to move across the manifold and out of local minima. Since CSGSA calculates the geodesic distance along the manifold, this biased information effect is less pronounced.

In some functions, such as F_{12} and F_{15}, CSGSA failed to converge to the global minimum. This may be due to the stochastic nature of the algorithm and happens also in the other algorithms. For example, ALO failed to converge for F_{13} and PSO failed in F_{27}. This phenomenon may be solved in future optimization of the hyper-parameters as we discuss later in the conclusions. In addition, it is important to note that in many cases, the difference between the converged values is very small (e.g., 5.64×10^{-6} between PSO and CSGSA in F_{22}), and insignificant in most practical applications. Thus, we can conclude that CSGSA, when observing the final fitness value, is at least comparable to the other algorithms and outperforms them in many cases.

In addition to the final fitness value attribute, the use of manifold learning can provide faster convergence. From the results, it is clear that CSGSA converges faster than the other algorithms in the majority of the functions. Figure 3 presents the convergence of the four algorithms on six selected functions. The functions were selected as they best reflect the convergence behaviour of CSGSA compared to the other three methods. The high convergence rate is due to the more fitted attraction forces provided through the manifold learning. This property is significant in high-frequency computations where efficiency is important. It is also interesting to understand the relative convergence rate of the different algorithms with various families of functions. In general, the common way of comparing stochastic algorithms is mainly based on experiments where the parameter of choice is the final fitness value or the rate of convergance. Table 13 provides a summary of the convergence rate of the different algorithms for various families of functions. Since we have built upon the implementation of GSA, the benefit of the manifold learning property is clear as it provides better convergence speed in the majority of the cases.

6. Conclusions

In this work we modified the original GSA by improving the information transfer through manifold learning and incorporated elitism. The manifold learning using Diffusion maps enabled the acquisition of more fitted forces between agents. The performance of the proposed algorithm was benchmarked and compared to other state-of-the-art algorithms over 47 test functions. The results show that the proposed algorithm can find better optima in many of the benchmark functions. Moreover, the manifold learning approach has been proven to be highly beneficial in terms of convergence rate.

The main weakness of metaheuristic algorithms is the vague understanding of what is the appropriate size of the population needed in order to find a good solution. Future work may include a systematic analysis to state-of-the-art algorithms, including the CSGSA, to reason about the

Table 7: Comparative results for the benchmark functions in Table 1 with population $N = 50$ and maximum number of iterations 1,000.

F		ALO	PSO	GSA	CSGSA
1	Average best-so-far	0.17	5.151×10^{-15}	3.144	7.65×10^{-11}
	Standard deviation	0.448	1.445×10^{-15}	1.931	2.6×10^{-10}
	p-value	1.734×10^{-6}	0.003	1.734×10^{-6}	
	Avg. num. iter. to convergence	925 ± 166	328 ± 172	86 ± 79	143 ± 17
2	Average best-so-far	0.032	0.013	0.016	0.007
	Standard deviation	0.018	0.011	0.009	0.004
	p-value	6.983×10^{-6}	0.062	8.918×10^{-5}	
	Avg. num. iter. to convergence	969 ± 26	140 ± 20	918 ± 53	261 ± 43
3	Average best-so-far	-2.062	-2.062	-2.062	-2.062
	Standard deviation	3.906×10^{-15}	9.033×10^{-16}	9.033×10^{-16}	2.272×10^{-9}
	p-value	8.77×10^{-5}	1.819×10^{-5}	1.819×10^{-5}	
	Avg. num. iter. to convergence	3 ± 1	1 ± 0	3 ± 2	2 ± 1
4	Average best-so-far	-0.955	-1	-0.994	-0.999
	Standard deviation	0.029	0	0.004	4.682×10^{-9}
	p-value	2.161×10^{-5}	5.588×10^{-6}	1.734×10^{-6}	
	Avg. num. iter. to convergence	39 ± 46	21 ± 10	124 ± 43	17 ± 7
5	Average best-so-far	-911.625	-860.355	-723.484	-945.531
	Standard deviation	84.912	117.063	127.909	31.495
	p-value	0.926	0.001	1.734×10^{-6}	
	Avg. num. iter. to convergence	19 ± 22	11 ± 10	1 ± 1	106 ± 152
6	Average best-so-far	-0.869	-0.869	-0.869	-
	Standard deviation	1.095×10^{-15}	5.646×10^{-16}	2.022×10^{-5}	-
	p-value	-	-	-	
	Avg. num. iter. to convergence	5 ± 3	2 ± 1	4 ± 3	-
7	Average best-so-far	0.216	0.088	0.032	-
	Standard deviation	0.097	0.034	0.056	-
	p-value	-	-	-	
	Avg. num. iter. to convergence	719 ± 47	154 ± 77	156 ± 98	-
8	Average best-so-far	-19.208	-19.208	-19.193	-19.208
	Standard deviation	5.643×10^{-13}	5.826×10^{-15}	0.023	3.135×10^{-5}
	p-value	2.353×10^{-6}	2.563×10^{-6}	0.02	
	Avg. num. iter. to convergence	33 ± 34	6 ± 4	18 ± 14	30 ± 20
9	Average best-so-far	-	-	-	-
	Standard deviation	-	-	-	-
	p-value	-	-	-	
	Avg. num. iter. to convergence	-	-	-	-
10	Average best-so-far	0.527	1.499×10^{-32}	5.789×10^{-19}	3.914×10^{-5}
	Standard deviation	0.599	1.113×10^{-47}	2.19×10^{-19}	1.084×10^{-4}
	p-value	2.613×10^{-4}	1.734×10^{-6}	1.734×10^{-6}	
	Avg. num. iter. to convergence	512 ± 357	246 ± 25	997 ± 4	372 ± 242
11	Average best-so-far	7.529×10^{-14}	1.349×10^{-31}	2.741×10^{-20}	1.779×10^{-30}
	Standard deviation	9.082×10^{-14}	6.68×10^{-47}	2.706×10^{-20}	3.715×10^{-30}
	p-value	0	0	0	
	Avg. num. iter. to convergence	976 ± 15	151 ± 3	989 ± 16	48 ± 48
12	Average best-so-far	17.212	10.38	3.051	4.416
	Standard deviation	7.813	4.454	1.63	3.287
	p-value	2.126×10^{-6}	2.596×10^{-5}	0.068	
	Avg. num. iter. to convergence	344 ± 79	86 ± 25	269 ± 53	283 ± 181
13	Average best-so-far	3.589×10^{-15}	0	0.007	0
	Standard deviation	3.139×10^{-15}	0	0.01	0
	p-value	3.759×10^{-6}	1	1.734×10^{-6}	
	Avg. num. iter. to convergence	909 ± 247	1 ± 0	18 ± 16	1 ± 0
14	Average best-so-far	0.5	0.5	0.5	0.5
	Standard deviation	1.498×10^{-8}	0	5.843×10^{-5}	2.803×10^{-6}
	p-value	1.734×10^{-6}	1.734×10^{-6}	1.734×10^{-6}	
	Avg. num. iter. to convergence	2 ± 0	1 ± 0	1 ± 0	1 ± 0
15	Average best-so-far	1657.469	1600.337	-	1311.05
	Standard deviation	617.404	275.922	-	263.918
	p-value	-	0	0	
	Avg. num. iter. to convergence	-	53 ± 7	0 ± 0	299 ± 192
16	Average best-so-far	-186.73	-186.73	-184.474	-186.73
	Standard deviation	9.363×10^{-11}	2.938×10^{-14}	2.864	6.384×10^{-5}
	p-value	1.149×10^{-4}	2.563×10^{-6}	1.92×10^{-6}	
	Avg. num. iter. to convergence	74 ± 40	16 ± 8	45 ± 29	38 ± 24

Table 8: Comparative results for the benchmark functions in Table 2 with population $N = 50$ and maximum number of iterations 1,000.

F		ALO	PSO	GSA	CSGSA
17	Average best-so-far	2.605×10^{-11}	0	0	0
	Standard deviation	3.57×10^{-11}	0	0	0
	p-value	1.734×10^{-6}	1	1	
	Avg. num. iter. to convergence	980 ± 14	1 ± 0	1 ± 0	1 ± 0
18	Average best-so-far	4.069	1.335	8.15	5.064
	Standard deviation	6.395	2.915	8.833	6.584
	p-value	0.416	0.024	0.051	
	Avg. num. iter. to convergence	716 ± 166	393 ± 372	378 ± 175	269 ± 217
19	Average best-so-far	1.214×10^{-8}	2.52×10^{-206}	5.998×10^{-18}	2.37×10^{-4}
	Standard deviation	1.079×10^{-8}	0	2.746×10^{-18}	4.009×10^{-4}
	p-value	1.734×10^{-6}	1.734×10^{-6}	1.734×10^{-6}	
	Avg. num. iter. to convergence	996 ± 5	1000 ± 0	997 ± 4	342 ± 192
20	Average best-so-far	4.226×10^{-12}	1.755×10^{-210}	1.716×10^{-18}	1.588×10^{-8}
	Standard deviation	1.797×10^{-12}	0	6.75×10^{-19}	2.64×10^{-8}
	p-value	1.734×10^{-6}	1.734×10^{-6}	1.734×10^{-6}	
	Avg. num. iter. to convergence	987 ± 10	1000 ± 0	997 ± 3	360 ± 215
21	Average best-so-far	2.788×10^{-8}	0	4.441×10^{-12}	6.379×10^{-16}
	Standard deviation	1.72×10^{-8}	0	5.564×10^{-12}	2.052×10^{-15}
	p-value	1.734×10^{-6}	1.734×10^{-6}	1.734×10^{-6}	
	Avg. num. iter. to convergence	886 ± 6	1 ± 0	758 ± 13	251 ± 215
22	Average best-so-far	3.826×10^{-10}	3.347×10^{-209}	5.838×10^{-18}	5.648×10^{-6}
	Standard deviation	6.756×10^{-10}	0	3.211×10^{-18}	1.456×10^{-5}
	p-value	1.734×10^{-6}	1.734×10^{-6}	1.734×10^{-6}	
	Avg. num. iter. to convergence	994 ± 7	1000 ± 0	997 ± 4	327 ± 173
23	Average best-so-far	-209.999	-209.999	-209.958	-147.396
	Standard deviation	7.904×10^{-10}	2.677×10^{-11}	0.086	41.581
	p-value	1.734×10^{-6}	1.734×10^{-6}	1.734×10^{-6}	
	Avg. num. iter. to convergence	149 ± 18	71 ± 13	189 ± 14	89 ± 101

Table 9: Comparative results for the benchmark functions in Table 3 with population $N = 50$ and maximum number of iterations 1,000.

F		ALO	PSO	GSA	CSGSA
24	Average best-so-far	1.861×10^{-14}	0	1.339×10^{-20}	6.864×10^{-13}
	Standard deviation	2.443×10^{-14}	0	1.405×10^{-20}	3.129×10^{-12}
	p-value	0.024	1.734×10^{-6}	7.712×10^{-4}	
	Avg. num. iter. to convergence	976 ± 20	1 ± 0	990 ± 23	287 ± 304
25	Average best-so-far	9.752×10^{-16}	0	7.538×10^{-22}	1.523×10^{-11}
	Standard deviation	1.365×10^{-15}	0	7.519×10^{-22}	3.828×10^{-11}
	p-value	0.001	1.734×10^{-6}	4.729×10^{-6}	
	Avg. num. iter. to convergence	976 ± 16	1 ± 0	989 ± 11	540 ± 294
26	Average best-so-far	-1.913	-1.913	-1.913	-1.913
	Standard deviation	4.778×10^{-15}	6.775×10^{-16}	6.775×10^{-16}	6.775×10^{-16}
	p-value	3.034×10^{-6}	1	1	
	Avg. num. iter. to convergence	4 ± 2	2 ± 1	9 ± 10	5 ± 1
27	Average best-so-far	-	-	-	-
	Standard deviation	-	-	-	-
	p-value	-	-	-	
	Avg. num. iter. to convergence	-	-	-	-
28	Average best-so-far	2.423×10^{-11}	5.049×10^{-64}	5.232×10^{-18}	0.104
	Standard deviation	1.635×10^{-11}	2.751×10^{-63}	2.334×10^{-18}	0.125
	p-value	1.734×10^{-6}	1.734×10^{-6}	1.734×10^{-6}	
	Avg. num. iter. to convergence	994 ± 6	1000 ± 1	994 ± 6	228 ± 194

Table 10: Comparative results for the benchmark functions in Table 4 with population $N = 50$ and maximum number of iterations 1,000.

F		ALO	PSO	GSA	CSGSA
29	Average best-so-far	1.551×10^{-15}	0	6.818×10^{-21}	9.696×10^{-25}
	Standard deviation	2.102×10^{-15}	0	7.797×10^{-21}	3.472×10^{-24}
	p-value	1.734×10^{-6}	1.734×10^{-6}	1.734×10^{-6}	
	Avg. num. iter. to convergence	977 ± 13	1 ± 0	989 ± 12	485 ± 236
30	Average best-so-far	-1.031	-1.031	-1.031	-1.031
	Standard deviation	9.32×10^{-15}	4.516×10^{-16}	4.516×10^{-16}	7.884×10^{-11}
	p-value	0.112	0.125	0.125	
	Avg. num. iter. to convergence	19 ± 19	3 ± 2	18 ± 17	5 ± 2
31	Average best-so-far	0.333	0.622	0.666	0.669
	Standard deviation	0.339	0.169	6.519×10^{-17}	0.005
	p-value	1.92×10^{-6}	1.734×10^{-6}	1.734×10^{-6}	
	Avg. num. iter. to convergence	705 ± 295	93 ± 61	219 ± 9	56 ± 37
32	Average best-so-far	13.655	1.536	5.392	8.402
	Standard deviation	31.104	1.287	0.132	0.535
	p-value	9.626×10^{-4}	1.734×10^{-6}	1.734×10^{-6}	
	Avg. num. iter. to convergence	800 ± 78	848 ± 156	499 ± 1	75 ± 74

Table 11: Comparative results for the benchmark functions in Table 5 with population $N = 50$ and maximum number of iterations 1,000.

F		ALO	PSO	GSA	CSGSA
33	Average best-so-far	1.395	2.021	3.769	1.436
	Standard deviation	0.669	1.381	2.904	0.614
	p-value	0.141	0.205	3.112×10^{-5}	
	Avg. num. iter. to convergence	67 ± 35	35 ± 24	22 ± 18	254 ± 173
34	Average best-so-far	-0.999	-1	-0.833	-0.966
	Standard deviation	4.399×10^{-12}	0	0.379	0.182
	p-value	3.112×10^{-5}	1	0.218	
	Avg. num. iter. to convergence	154 ± 41	32 ± 8	66 ± 33	27 ± 6
35	Average best-so-far	-6.688	-8.776	-9.289	-9.008
	Standard deviation	1.126	0.553	0.174	0.523
	p-value	2.126×10^{-6}	0.171	0.049	
	Avg. num. iter. to convergence	255 ± 54	67 ± 22	251 ± 11	132 ± 86

Table 12: Comparative results for the benchmark functions in Table 6 with population $N = 50$ and maximum number of iterations 1,000.

F		ALO	PSO	GSA	CSGSA
36	Average best-so-far	0.076	0.076	8.247×10^{-21}	9.763×10^{-8}
	Standard deviation	0.232	0.232	8.917×10^{-21}	5.162×10^{-7}
	p-value	0.044	0.002	1.92×10^{-6}	
	Avg. num. iter. to convergence	879 ± 288	3 ± 7	985 ± 24	187 ± 113
37	Average best-so-far	0.397	0.397	0.123	0.397
	Standard deviation	2.89×10^{-14}	0	0.187	2.318×10^{-9}
	p-value	0	0	0	
	Avg. num. iter. to convergence	77 ± 37	9 ± 5	27 ± 45	31 ± 14
38	Average best-so-far	1.053	1.269×10^{-4}	1.232	0.712
	Standard deviation	2.384	1.362×10^{-4}	1.589	1.044
	p-value	0.428	1.734×10^{-6}	0.452	
	Avg. num. iter. to convergence	844 ± 244	994 ± 6	543 ± 162	312 ± 193
39	Average best-so-far	-6.02	-6.02	-6.02	-
	Standard deviation	3.282×10^{-15}	2.71×10^{-15}	3.486×10^{-7}	-
	p-value	-	-	-	
	Avg. num. iter. to convergence	2 ± 1	1 ± 0	1 ± 0	-
40	Average best-so-far	3	2.999	2.999	2.999
	Standard deviation	1.77×10^{-13}	8.287×10^{-16}	2.008×10^{-15}	8.53×10^{-16}
	p-value	1.723×10^{-6}	0.289	1.242×10^{-4}	
	Avg. num. iter. to convergence	35 ± 30	9 ± 5	154 ± 51	10 ± 1
41	Average best-so-far	-3.862	-3.862	-3.862	-3.862
	Standard deviation	9.429×10^{-15}	3.161×10^{-15}	3.161×10^{-15}	3.062×10^{-15}
	p-value	1.893×10^{-6}	1	1	
	Avg. num. iter. to convergence	7 ± 7	3 ± 1	15 ± 13	4 ± 2
42	Average best-so-far	-3.126	-3.094	-3.134	-3.134
	Standard deviation	0.043	0.09	4.516×10^{-16}	4.087×10^{-13}
	p-value	0.044	0.791	4.882×10^{-4}	
	Avg. num. iter. to convergence	68 ± 32	4 ± 2	131 ± 59	11 ± 4
43	Average best-so-far	-3.005	-3.026	-3.042	-3.042
	Standard deviation	0.03	0.027	1.355×10^{-15}	1.069×10^{-8}
	p-value	5.086×10^{-4}	0.909	1.227×10^{-5}	
	Avg. num. iter. to convergence	79 ± 35	6 ± 2	198 ± 7	19 ± 6
44	Average best-so-far	6.617×10^{14}	1.053×10^{12}	1.367×10^{16}	2.586×10^{14}
	Standard deviation	1.767×10^{15}	3.705×10^{12}	2.204×10^{16}	7.185×10^{14}
	p-value	0.813	1.734×10^{-6}	1.92×10^{-6}	
	Avg. num. iter. to convergence	678 ± 209	940 ± 71	96 ± 147	347 ± 134
45	Average best-so-far	4.956×10^{-4}	4.981×10^{-7}	6.439×10^{-5}	0.004
	Standard deviation	2.998×10^{-4}	5.621×10^{-7}	5.455×10^{-5}	0.004
	p-value	1.493×10^{-5}	1.734×10^{-6}	1.734×10^{-6}	
	Avg. num. iter. to convergence	922 ± 9	951 ± 54	712 ± 17	316 ± 211
46	Average best-so-far	-7.558	-7.395	-6.644	-5.341
	Standard deviation	3.054	3.445	2.482	2.368
	p-value	0.005	0.025	0.008	
	Avg. num. iter. to convergence	121 ± 16	31 ± 6	117 ± 48	143 ± 197
47	Average best-so-far	-361.032	-348.309	-177.279	-389.508
	Standard deviation	26.534	15.721	192.849	5.577
	p-value	1.36×10^{-5}	1.734×10^{-6}	1.024×10^{-5}	
	Avg. num. iter. to convergence	116 ± 8	18 ± 3	57 ± 61	73 ± 55

Table 13: Percentage in which CSGSA performs equally or better than the compared methods in terms of the converged fitness value and rate of convergence.

	CSGSA vs. ALO		CSGSA vs. PSO		CSGSA vs. GSA	
	fitness (%)	conv. (%)	fitness (%)	conv. (%)	fitness (%)	conv. (%)
Many Local minima functions (1-16)	69	100	54	46	83	67
Bowl shaped functions (17-23)	43	100	14	57	57	100
Plate shaped functions (24-28)	60	100	40	60	40	100
Valley shaped functions (29-32)	50	100	25	75	50	100
Step Ridges/Drops (33-35)	100	67	100	33	67	67
Other functions (36-47)	75	100	58	50	58	92

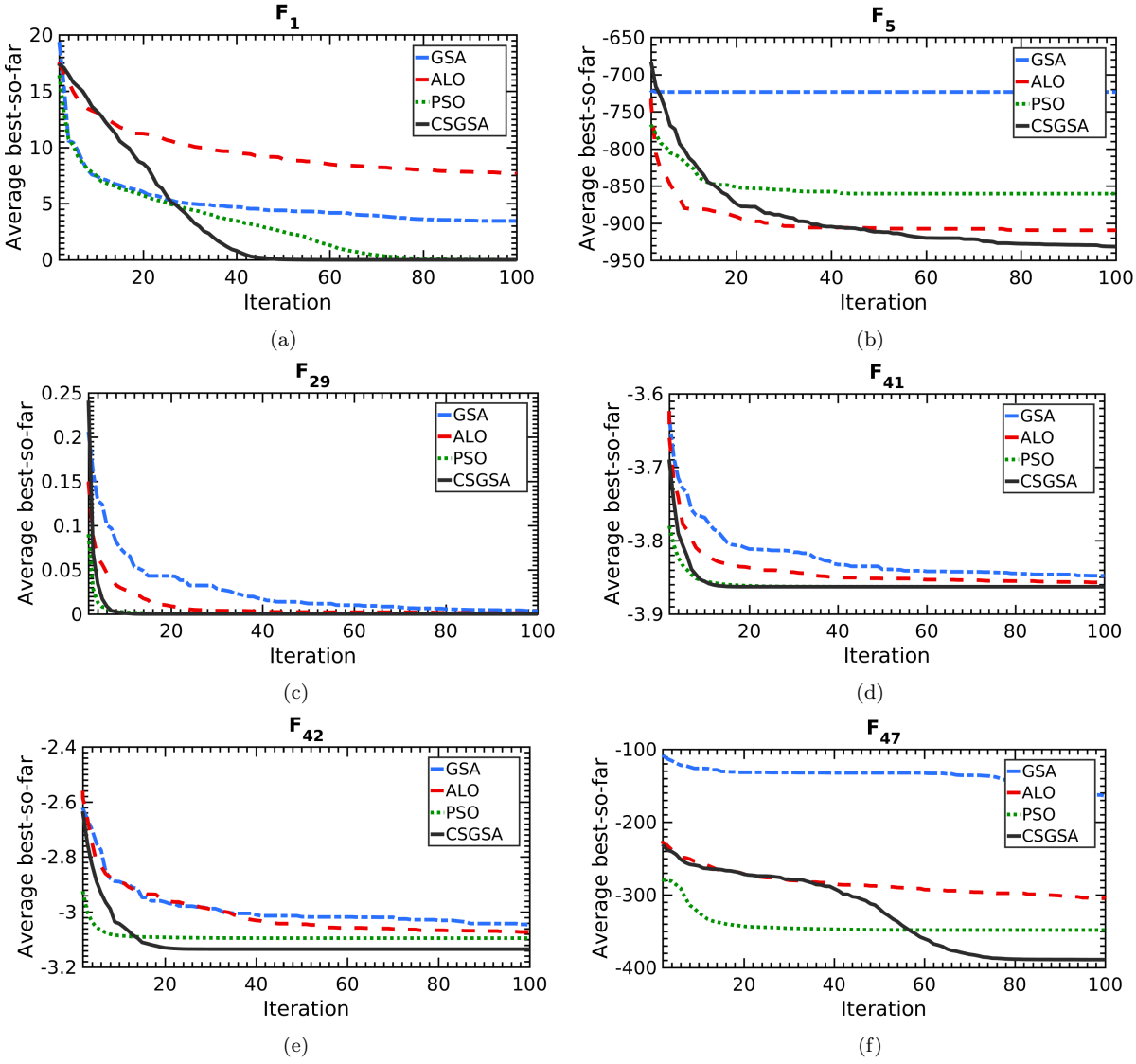


Figure 3: Comparison performance of GSA, ALO, PSO and CSGSA for minimization of (a) F_1, (b) F_5, (c) F_{29}, (d) F_{41}, (e) F_{42} and (f) F_{47}.

Table A.1: The 47 benchmark functions.

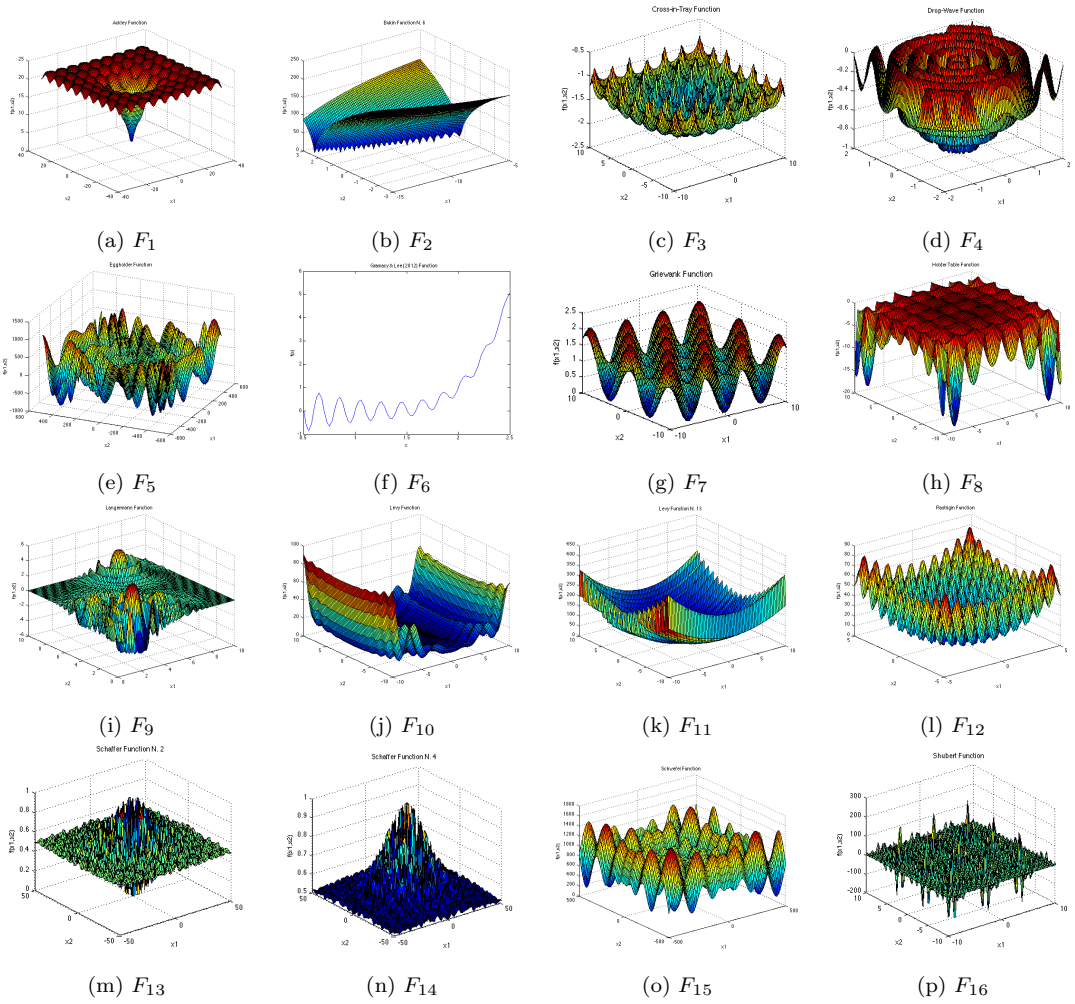
Function Index	Function Name	Function Index	Function Name
	Many Local Minima Functions		
1	Ackley Function	2	Bukin Function N.6
3	Cross-In-Tray Function	4	Drop-Wave Function
5	Eggholder Function	6	Gramacy and Lee (2012) Function
7	Griewank Function	8	Holder Table Function
9	Langermann Function	10	Levy Function
11	Levy Function N.13	12	Rastrigin Function
13	Schaffer Function N.2	14	Schaffer Function N.4
15	Schwefel Function	16	Shubert Function
	Bowl-Shaped Functions		
17	Bohachevsky Functions	18	Perm Function 0,d, β
19	Rotated Hyper-Ellipsoid Function	20	Sphere Function
21	Sum Of Different Powers Function	22	Sum Squares Function
23	Trid Function		
	Plate-Shaped Functions		
24	Booth Function	25	Matyas Function
26	McCormick Function	27	Power Sum Function
28	Zakharov Function		
	Valley-Shaped Functions		
29	Three-Hump Camel Function	30	Six-Hump Camel Function
31	Dixon-Price Function	32	Rosenbrock Function
	Steep Ridges/Drops Functions		
33	De Jong Function N.5	34	Easom Function
35	Michalewicz Function		
	Other Functions		
36	Beale Function	37	Branin Function
38	Colville Function	39	Forrester et al. (2008) Function
40	Goldstein-Price Function	41	Hartmann 3-Dimensional Function
42	Hartmann 4-Dimensional Function	43	Hartmann 6-Dimensional Function
44	Perm Function d, β	45	Powell Function
46	Shekel Function	47	Styblinski-Tang Function

effect of the population size on the solution and convergence rate. Also, the choice of the hyper-parameters for all mentioned meta-heuristic algorithms has a large effect on performance and they should be optimized.

Appendix A.

Table A.2: Additional information for some benchmark functions.

Func.	Additional Information
F_1	Recommended variable values are $a = 20$, $b = 0.2$ and $c = 2\pi$
F_9	$For\ d = 2 : m = 5,\ c = (1, 2, 5, 2, 3),\ A = \begin{pmatrix} 3 & 5 \\ 5 & 2 \\ 2 & 1 \\ 1 & 4 \\ 7 & 9 \end{pmatrix}$
F_{10}	$w_i = 1 + \frac{x_i - 1}{4}, for\ all\ i = 1, ..., d$
F_{27}	The recommended value of the b-vector, for $d = 4$, is: $b = (8, 18, 44, 114)$
F_{33}	$a = \begin{pmatrix} -32 & -16 & 0 & 16 & 32 & -32 & ... & 0 & 16 & 32 \\ -32 & -32 & -32 & -32 & -32 & -16 & ... & 32 & 32 & 32 \end{pmatrix}$
F_{35}	The recommended value of m is $m = 10$
F_{37}	The recommended values are $a = 1$, $b = \frac{5.1}{4\pi^2}$, $c = \frac{5}{\pi}$, $r = 6$, $s = 10$ and $t = \frac{1}{8\pi}$
F_{41}	$\alpha = (1.0, 1.2, 3.0, 3.2)^T,\ A = \begin{pmatrix} 3.0 & 10 & 30 \\ 0.1 & 10 & 35 \\ 3.0 & 10 & 30 \\ 0.1 & 10 & 35 \end{pmatrix},\ P = 10^{-4} \begin{pmatrix} 3689 & 1170 & 2673 \\ 4699 & 4387 & 7470 \\ 1091 & 8732 & 5547 \\ 381 & 5743 & 8828 \end{pmatrix}$
F_{42}	$\alpha = (1.0, 1.2, 3.0, 3.2)^T,\ A = \begin{pmatrix} 10 & 3 & 17 & 3.50 & 1.7 & 8 \\ 0.05 & 10 & 17 & 0.1 & 8 & 14 \\ 3 & 3.5 & 1.7 & 10 & 17 & 8 \\ 17 & 8 & 0.05 & 10 & 0.1 & 14 \end{pmatrix},\ P = 10^{-4} \begin{pmatrix} 1312 & 1696 & 5569 & 124 & 8283 & 5886 \\ 2329 & 4135 & 8307 & 3736 & 1004 & 9991 \\ 2348 & 1451 & 3522 & 2883 & 3047 & 6650 \\ 4047 & 8828 & 8732 & 5743 & 1091 & 381 \end{pmatrix}$
F_{43}	$\alpha = (1.0, 1.2, 3.0, 3.2)^T,\ A = \begin{pmatrix} 10 & 3 & 17 & 3.50 & 1.7 & 8 \\ 0.05 & 10 & 17 & 0.1 & 8 & 14 \\ 3 & 3.5 & 1.7 & 10 & 17 & 8 \\ 17 & 8 & 0.05 & 10 & 0.1 & 14 \end{pmatrix},\ P = 10^{-4} \begin{pmatrix} 1312 & 1696 & 5569 & 124 & 8283 & 5886 \\ 2329 & 4135 & 8307 & 3736 & 1004 & 9991 \\ 2348 & 1451 & 3522 & 2883 & 3047 & 6650 \\ 4047 & 8828 & 8732 & 5743 & 1091 & 381 \end{pmatrix}$
F_{46}	$m = 10,\ \beta = \frac{1}{10}(1, 2, 2, 4, 4, 6, 3, 7, 5, 5)^T,\ C = \begin{pmatrix} 4.0 & 1.0 & 8.0 & 6.0 & 3.0 & 2.0 & 5.0 & 8.0 & 6.0 & 7.0 \\ 4.0 & 1.0 & 8.0 & 6.0 & 7.0 & 9.0 & 3.0 & 1.0 & 2.0 & 3.6 \\ 4.0 & 1.0 & 8.0 & 6.0 & 3.0 & 2.0 & 5.0 & 8.0 & 6.0 & 7.0 \\ 4.0 & 1.0 & 8.0 & 6.0 & 7.0 & 9.0 & 3.0 & 1.0 & 2.0 & 3.6 \end{pmatrix}$



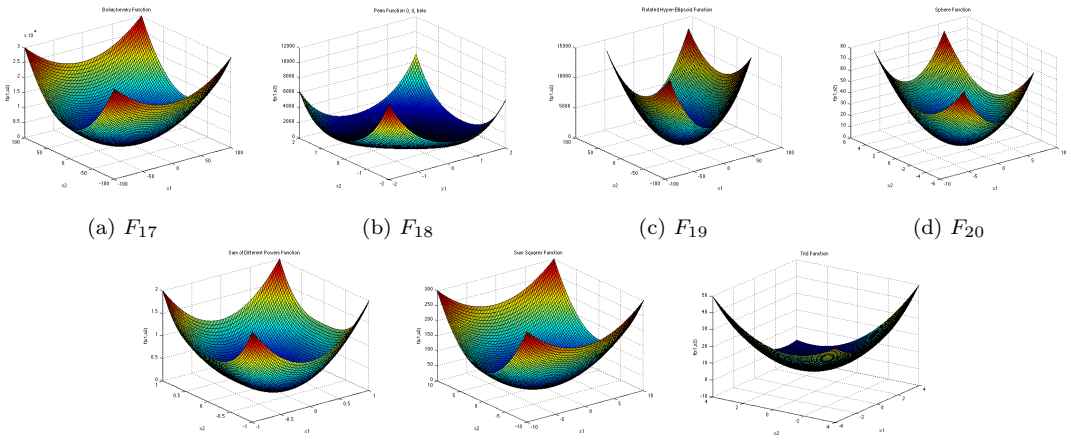
(a) F_1 (b) F_2 (c) F_3 (d) F_4

(e) F_5 (f) F_6 (g) F_7 (h) F_8

(i) F_9 (j) F_{10} (k) F_{11} (l) F_{12}

(m) F_{13} (n) F_{14} (o) F_{15} (p) F_{16}

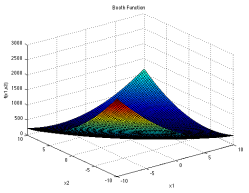
Figure A.1: Many local minima functions [32].



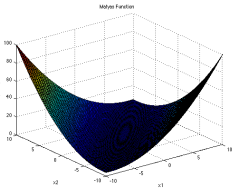
(a) F_{17} (b) F_{18} (c) F_{19} (d) F_{20}

(e) F_{21} (f) F_{22} (g) F_{23}

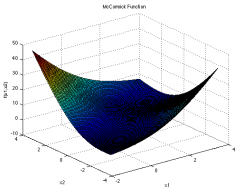
Figure A.2: Bowl-shaped functions [32].



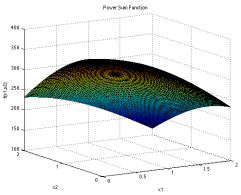
(a) F_{24}



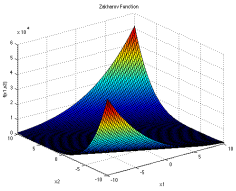
(b) F_{25}



(c) F_{26}

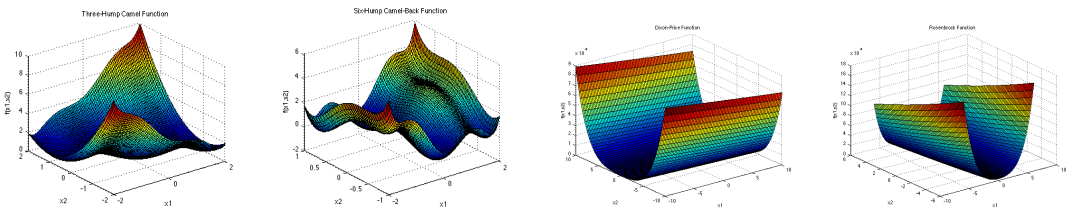


(d) F_{27}



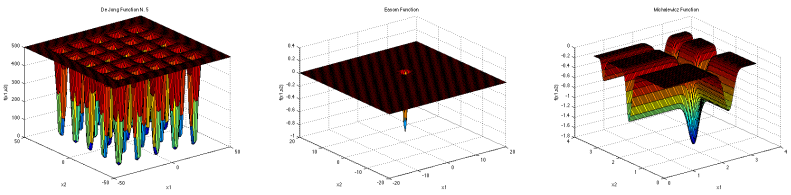
(e) F_{28}

Figure A.3: Plate-shaped functions [32].



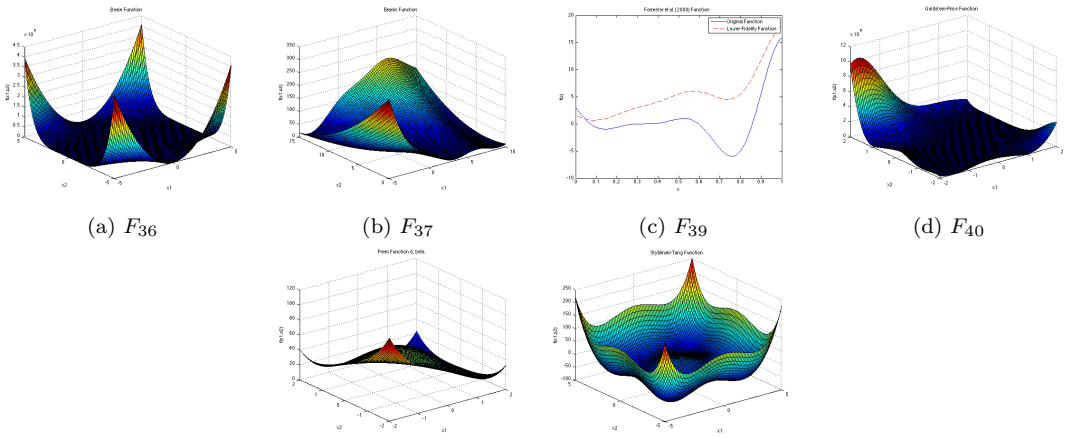
(a) F_{29} (b) F_{30} (c) F_{31} (d) F_{32}

Figure A.4: Valley-shaped functions [32].



(a) F_{33} (b) F_{34} (c) F_{35}

Figure A.5: Steep ridges/drops functions [32].



(a) F_{36} (b) F_{37} (c) F_{39} (d) F_{40}

(e) F_{44} (f) F_{47}

Figure A.6: Other functions [32].

References

[1] Baldi, P., Sadowski, P., Whiteson, D., 2014. Searching for exotic particles in high-energy physics with deep learning. Nature communications 5, 4308.

[2] Belkin, M., Niyogi, P., 2003. Laplacian eigenmaps for dimensionality reduction and data representation. Neural computation 15, 1373–1396.

[3] Belkin, M., Niyogi, P., 2004. Semi-supervised learning on riemannian manifolds. Machine learning 56, 209–239.

[4] Bellman, R., 1954. The theory of dynamic programming. Technical Report. The RAND Corp.

[5] Van den Bergh, F., Engelbrecht, A.P., 2006. A study of particle swarm optimization particle trajectories. Information sciences 176, 937–971.

[6] Blum, C., Roli, A., 2003. Metaheuristics in combinatorial optimization: Overview and conceptual comparison. ACM computing surveys (CSUR) 35, 268–308.

[7] Chapelle, O., Schlkopf, B., Zien, A., 2010. Semi-Supervised Learning. 1st ed., The MIT Press.

[8] Chen, J., Xin, B., Peng, Z., Dou, L., Zhang, J., 2009. Optimal contraction theorem for exploration-exploitation tradeoff in search and optimization. IEEE Transactions on Systems, Man, and Cybernetics - Part A: Systems and Humans 39, 680–691.

[9] Coifman, R.R., Lafon, S., 2006. Diffusion maps. Applied and computational harmonic analysis 21, 5–30.

[10] Doraghinejad, M., Nezamabadi-pour, H., 2014. Black hole: A new operator for gravitational search algorithm. International Journal of Computational Intelligence Systems 7, 809–826.

[11] Dorigo, M., Maniezzo, V., Colorni, A., 1996. Ant system: optimization by a colony of cooperating agents. IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics) 26, 29–41.

[12] Gepshtein, S., Keller, Y., 2013. Image completion by diffusion maps and spectral relaxation. IEEE Transactions on Image Processing 22, 2983–2994.

[13] Güvenç, U., Katırcıoğlu, F., 2017. Escape velocity: a new operator for gravitational search algorithm. Neural Computing and Applications , 1–16.

[14] Hereid, A., Cousineau, E.A., Hubicki, C.M., Ames, A.D., 2016. 3D dynamic walking with underactuated humanoid robots: A direct collocation framework for optimizing hybrid zero dynamics, in: Proceeding of the IEEE International Conference on Robotics and Automation (ICRA), pp. 1447–1454.

[15] Holland, J., 1992. Genetic algorithms. Scientific American 267, 66–72.

[16] Karaboga, D., Basturk, B., 2007. A powerful and efficient algorithm for numerical function optimization: artificial bee colony (ABC) algorithm. Journal of Global Optimization 39, 459–471.

[17] Kaveh, A., Bakhshpoori, T., 2016. Water evaporation optimization: A novel physically inspired optimization algorithm. Computers & Structures 167, 69–85.

[18] Kennedy, J., 2011. Particle swarm optimization, in: Encyclopedia of machine learning. Springer, pp. 760–766.

[19] Kirkpatrick, S., Gelatt, C.D., Vecchi, M.P., et al., 1983. Optimization by simulated annealing. science 220, 671–680.

[20] Lafon, S., Lee, A.B., 2006. Diffusion maps and coarse-graining: a unified framework for dimensionality reduction, graph partitioning, and data set parameterization. IEEE Transactions on Pattern Analysis and Machine Intelligence 28, 1393–1403.

[21] Lande, R., 1988. Genetics and demography in biological conservation. Science (Washington) 241, 1455–1460.

[22] Liu, B., Zhang, Q., Gielen, G.G., 2013. A gaussian process surrogate model assisted evolutionary algorithm for medium scale expensive optimization problems. IEEE Transactions on Evolutionary Computation 18, 180–192.

[23] Medjahed, S.A., Saadi, T.A., Benyettou, A., Ouali, M., 2016. Gray wolf optimizer for hyperspectral band selection. Applied Soft Computing 40, 178–186.

[24] Mirjalili, S., 2015. The ant lion optimizer. Advances in Engineering Software 83, 80–98.

[25] Mirjalili, S., 2016. Dragonfly algorithm: a new meta-heuristic optimization technique for solving single-objective, discrete, and multi-objective problems. Neural Computing and Applications 27, 1053–1073.

[26] Mitchell, M., 1996. An Introduction to Genetic Algorithms. MIT Press, Cambridge, MA, USA.

[27] Narimani, M.R., Vahed, A.A., Azizipanah-Abarghooee, R., Javidsharifi, M., 2014. Enhanced gravitational search algorithm for multi-objective distribution feeder reconfiguration considering reliability, loss and operational cost. IET Generation, Transmission & Distribution 8, 55–69.

[28] Passino, K.M., 2010. Bacterial foraging optimization. Int. J. Swarm. Intell. Res. 1, 1–16.

[29] Porte, J.D.L., Herbst, B.M., Hereman, W., Walt, S.J.V.D., 2008. An introduction to diffusion maps, in: In The 19th Symposium of the Pattern Recognition Association of South Africa.

[30] Rashedi, E., Nezamabadi-Pour, H., Saryazdi, S., 2009. GSA: A gravitational search algorithm. Information sciences 179, 2232–2248.

[31] Sindhwani, V., Belkin, M., Niyogi, P., 2010. The geometric basis of semi-supervised learning. MIT press .

[32] Surjanovic, S., Bingham, D., 2017. Virtual library of simulation experiments: Test functions and datasets. `http://www.sfu.ca/s̃surjano`.

[33] Tenenbaum, J.B., De Silva, V., Langford, J.C., 2000. A global geometric framework for nonlinear dimensionality reduction. science 290, 2319–2323.

[34] Yang, X.S., 2010. Firefly algorithm, lévy flights and global optimization, in: Bramer, M., Ellis, R., Petridis, M. (Eds.), Research and Development in Intelligent Systems XXVI, Springer London, London. pp. 209–218.