Cluster Computing (2025)28:596
https://doi.org/10.1007/s10586-025-05265-5

Star Death: a novel lightweight metaheuristic algorithm and its application for dynamic load-balancing in cluster computing

Sasan Harifi[1] · Reza Eghbali[1] · Seyed Mohsen Mirhosseini[1]

Received: 25 October 2024 / Revised: 6 March 2025 / Accepted: 9 March 2025

Abstract
Optimization is a crucial principle in today's world, applied in various fields to increase profit and efficiency while reducing cost and time. However, solving optimization problems can be challenging, especially in dynamic environments where conditions are constantly changing. In the meantime, metaheuristic methods are effective for solving large and complex optimization problems. Due to the presentation of several algorithms in the last two decades, each of which has high complexity and is difficult to understand, providing a lightweight algorithm has become a principle. This paper proposes a novel lightweight metaheuristic algorithm called the Star Death (SD) algorithm, which is inspired by the physical process of star death. The proposed algorithm aims to model the exact, regular, and optimal physical process of star death that can solve various problems. For this purpose, the SD algorithm employs an elite strategy that dynamically adjusts the range of exploration for better solutions. It also uses center-based sampling that emphasizes the center point's proximity to solutions, enhancing the optimizer's effectiveness. In this algorithm, the parameters are adjusted adaptively to enhance clarity and understanding of the parameter space. To prove application and robustness, the SD algorithm has been compared with 10 standard and popular metaheuristic algorithms. Based on this, 45 different benchmark test functions have been used. In addition, the algorithm has been tested and evaluated in high dimensions space. Also, it has been applied to 57 real-world CEC 2020 problems and six classic engineering problems. As a specific application, the SD algorithm is also used in solving the dynamic load-balancing problem. The results are generally indicative of the potential of the proposed algorithm to effectively solve complex optimization problems. The source codes of the SD algorithm are publicly available at https://github.com/harifi/SD.

Keywords Metaheuristic · Optimization · Star Death (SD) algorithm · Physics-based algorithm · High-dimensional tests · Lightweight metaheuristic · Engineering problems · Dynamic load-balancing

1 Introduction

The global landscape is experiencing heightened complexity daily. The allocation of resources is constrained, emphasizing the critical need for their efficient utilization. The quest for effective and optimal problem-solving strategies in intricate scenarios necessitates the application of pragmatic methodologies. Over the past few decades, numerous optimization techniques have been introduced, offering versatile applications across a spectrum of optimization challenges [1] and yielding varying degrees of performance. Diverse factors, such as the characteristics of search spaces, can exert a notable influence on outcomes. Within the dichotomy of optimization methodologies—comprising deterministic and stochastic approaches—stochastic optimization methods exhibit enhanced efficacy in addressing large, intricate problems compared to deterministic counterparts. Nonetheless, stochastic optimization methods encounter challenges in highly complex scenarios, including issues related to run time, convergence towards local optima, and reliance on the nature of search spaces.

✉ Sasan Harifi
s.harifi@kiau.ac.ir

Reza Eghbali
std_reza.eghbali@khu.ac.ir

Seyed Mohsen Mirhosseini
m_mirhosseini@sbu.ac.ir

[1] Department of Computer Engineering, Karaj Branch, Islamic Azad University, Karaj, Iran

Published online: 30 August 2025

In the real world, many problems occur in dynamic environments, which is one of the most general types of non-deterministic problems. Unlike static environments, dynamic environments are always changing, and as a result, the position and amount of optimal points also change. Therefore, the solution methods must have the ability to adapt to environmental changes. Although exact methods can solve these optimization problems, these methods need a lot of time to solve the problem. Sometimes it is enough to reach the near-optimal solution and deal with the dynamic environment in a much shorter time. So, approximate algorithms can be used for this. Approximate algorithms can be divided into two categories: heuristic and metaheuristic. Heuristic methods are completely dependent on the type of problem so they are designed for some specific problems and are used for the same specific problem. In this way, a heuristic method specific to that problem should be designed for each problem. Metaheuristic methods are more general and common methods [2]. The diversity of their use in different problems is more than heuristic methods, so it can be said that they can be used for almost any type of problem. They effectively search the problem space and reduce the problem-solving time. In this way, they can be used even for very large problems.

Metaheuristics can be considered a subset of optimization in computer science and applied mathematics. They involve complex computing theory and algorithms. They are also used in other fields such as artificial intelligence, computational intelligence, and soft computing. Metaheuristics have been very effective and efficient in solving complex problems in the real world, so their role in reducing calculations and costs should not be denied [3]. Metaheuristic algorithms represent strategies crafted to efficiently address computationally challenging optimization problems. Researchers have drawn insights from a variety of natural and physical processes to devise metaheuristics that have effectively delivered near-optimal or optimal solutions for numerous engineering applications. The practical significance of metaheuristic algorithms has been widely acknowledged, particularly in recent years, owing to their speed, high-quality solutions, and problem-agnostic nature. However, no single metaheuristic can universally tackle all types of optimization challenges. Consequently, numerous metaheuristics have been introduced over time, to identify efficient metaheuristics suitable for diverse optimization problem categories. Notably, the design of metaheuristics hinges on mimicking the advancement or locomotion patterns of specific phenomena or organisms. By replicating such advancement or locomotion styles, a metaheuristic can explore the search space of a problem akin to the habitat of the emulated phenomenon or organism. Metaheuristics rely on two fundamental search strategies in their quest to identify the optimal solution for a given problem [2]. The initial strategy involves exploration, which delves into uncharted search regions. The subsequent strategy is exploitation, which scrutinizes the surroundings of the identified optimal solution. The optimal performance of any metaheuristic hinges on striking a balance between these two strategies. Notably, an excessive emphasis on exploration may hinder metaheuristics from reaching the globally optimal solution, whereas an overemphasis on exploitation could result in being trapped in local optima [4]. If we want to present a category of these algorithms, we can refer to Fig. 1. Even though, we cannot consider the classification in Fig. 1 to be unique because different classifications have been presented by authors. However, the classification presented in the figure seems to be more logical. This figure shows that algorithms fall into four categories. These four categories are Evolutionary-based, Trajectory-based, Ancient-inspired, and Nature-inspired [4].

Evolutionary algorithms simulate the idea of biological evolution. In this category, a population is considered a solution candidate population and constantly tries to change its genetic diversity. The concept of competition creates evolution. In this type of algorithm, a population is randomly selected so that each individual is a solution. Then, in successive iterations, competent individuals are selected and try to create a new population or a new generation of offspring [5]. Their children can achieve evolution and repeat the same process. Among the popular and common algorithms that can be included in this category are Genetic Algorithm (GA) [6], Memetic Algorithm (MA) [7], Differential Evolution (DE) [8], Harmony Search (HS) [9], Clonal Selection Algorithm (CSA) [10], Backtracking Search Algorithm (BSA) [11], Stochastic Fractal Search (SFS) [12], Across Neighborhood Search (ANS) [13], and so on.

Trajectory-based algorithms usually try to focus on one solution and improve it. They do this with an iterative routine so that through these iterative routines they are transferred from one solution space to another solution space. This group of algorithms can be very effective in some problems, especially problems that have a type of permutating state. The most popular algorithms of this category are Simulated Annealing (SA) [14], Tabu Search (TS) [15], Variable neighborhood search (VNS) [16], Guided Local Search (GLS) [17], and Iterative Local Search (ILS) [18].

The inspiration from the ancient is the source of the new inspiration that has been introduced recently. Various and complex human-made structures in ancient times and their creation mechanisms show a kind of optimization despite the many limitations that existed in that era [19]. The

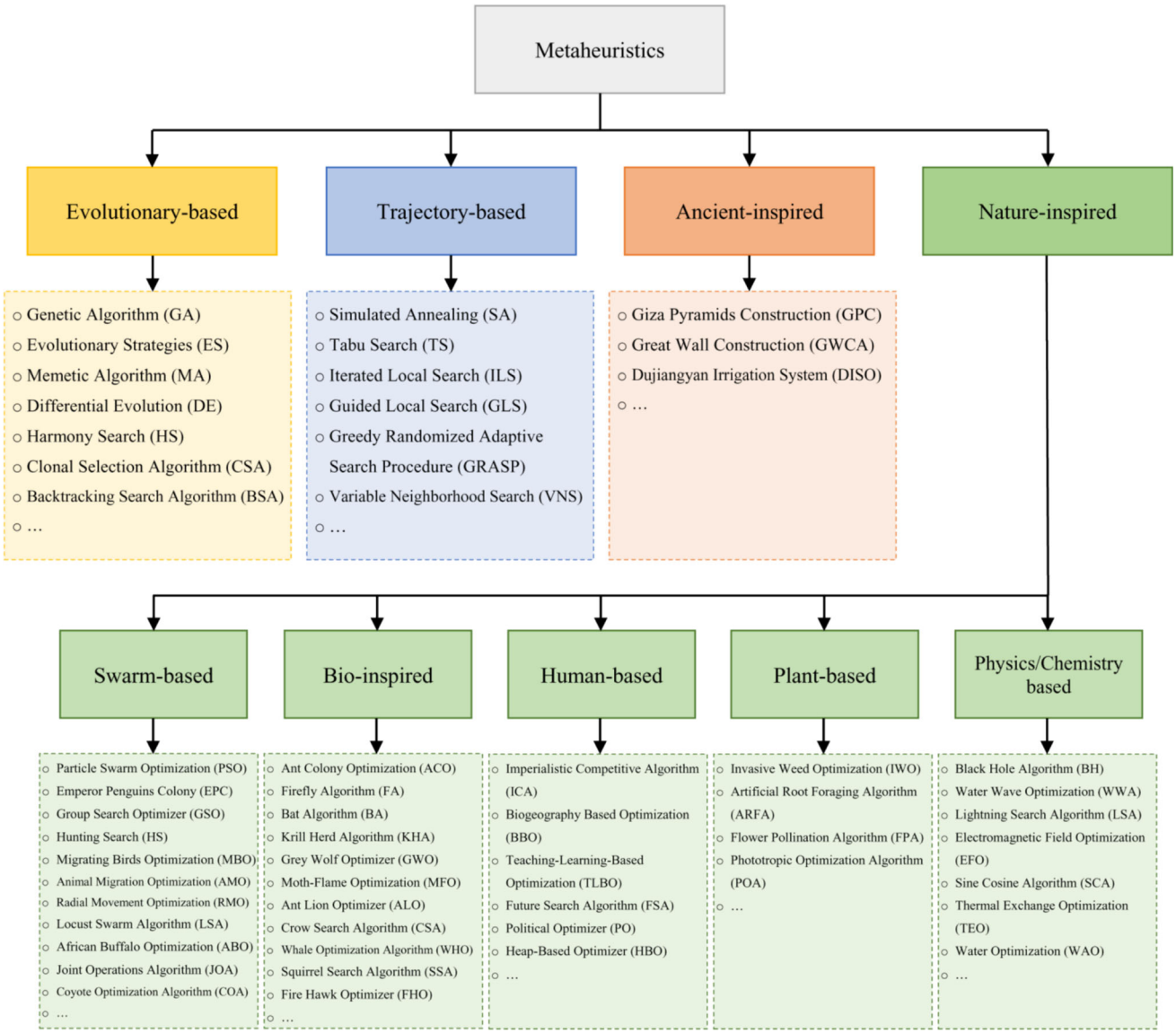

Fig. 1 Classification of metaheuristic algorithms

technology used in ancient civilizations was far ahead of its time. A closer look at these technologies can reveal the truths of optimization in ancient times. This category has paved the way for modern optimization techniques. Important algorithms of this category can be Giza Pyramids Construction (GPC) [4], the Great Wall Construction Algorithm (GWCA) which draws inspiration from the historical practice of the construction of the Great Wall [20], and Dujiangyan Irrigation System Optimization (DISO) [21]. These ancient-inspired algorithms offer simplicity, robustness, and competitive performance in solving complex optimization problems. Also, the GPC, for instance, utilizes a unique approach based on labor movement dynamics, showcasing efficiency and high convergence performance in comparison to existing methods. By combining traditional wisdom with modern computational concepts, these ancient metaheuristic algorithms continue to contribute significantly to the optimization field, offering innovative solutions to challenging problems [19].

One of the most popular categories is the nature-based category. Nature had billions of years to create, revise, and edit various types of creatures to adapt to itself. Nature had so much time that it was able to provide a solution to face any challenge. These solutions are the solutions that humans need to solve many of their engineering problems. Nature has simple and understandable rules [22]. This category is so large that it can itself include subcategories such as Swarm-based, Bio-inspired, Human-based, Plant-based, and Physics/chemistry-based.

Swarm-based subcategory can be considered swarm intelligence or group and collective behavior related to a group of animals. In computing applications, swarm intelligence is modeled on organisms or groups such as ants, bees, fish, birds, and so on. In this type of community, each of the agents or entities has a relatively simple structure, but their swarm behavior seems complicated. In other words, there is a very complex relationship between swarm behavior and the individual behavior of a community. Swarm behavior is not only dependent on the individual behavior of the agents and members of the community but also on the way of interaction between individuals. Some of the most important and popular algorithms in this subcategory are Particle Swarm Optimization (PSO) [23], Glowworm Swarm Optimization (GSO) [24], Intelligent Water Drops (IWD) [25], Group Search Optimizer (GSO) [26], Hunting Search (HS) [27], Migrating Birds Optimization (MBO) [28], Animal Migration Optimization (AMO) [29], Radial Movement Optimization (RMO) [30], Locust Swarm Algorithm (LSA) [31], African Buffalo Optimization (ABO) [32], Joint Operations Algorithm (JOA) [33], Coyote Optimization Algorithm (COA) [34], Emperor Penguins Colony (EPC) [35], and Special Forces Algorithm (SFA) [36].

Swarm-based algorithms are sometimes considered a subcategory of bio-inspired algorithms. But it should also be noted that many bio-inspired algorithms do not directly use swarm behavior. For this reason, bio-inspired algorithms can be placed in a separate subcategory. The number of algorithms in this subcategory is very large. Some of them are Artificial Bee Colony (ABC) [37], Ant Colony Optimization (ACO) [38], Firefly Algorithm (FA) [39], Bat Algorithm (BA) [40], Krill Herd Algorithm (KHA) [41], Gray Wolf Optimizer (GWO) [42], Moth-Flame Optimization (MFO) [43], Ant Lion Optimizer (ALO) [44], Crow Search Algorithm (CSA) [45], Whale Optimization Algorithm (WHO) [46], Squirrel Search Algorithm (SSA) [47], Fire Hawk Optimizer (FHO) [48], GOOSE algorithm [49], Golden Jackal Optimization (GJO) [50], White Shark Optimizer (WSO) [51], Spider Wasp Optimization (SWO) [52], Puma Optimizer (PO) [53], Walrus Optimizer (WO) [54], and Flying Fox Optimization (FFO) [55].

Human-based subcategory models human behaviors. These behaviors include how humans search in the environment and how they behave in the environment. Also, social behaviors such as people's cooperation with each other can be modeled. Algorithms such as Imperialistic Competitive Algorithm (ICA) [56], Biogeography Based Optimization (BBO) [57], Teaching–Learning-Based Optimization (TLBO) [58], Future Search Algorithm (FSA) [59], Political Optimizer (PO) [60], Heap-Based Optimizer (HBO) [61], and Squid Game Optimizer (SGO) [62] can be placed in this subcategory.

In general, any algorithm that models the behavior of a plant is placed in the plant-based subcategory. Plant growth, plant seed dispersal, root growth, and so on can be modeled. Algorithms such as Invasive Weed Optimization (IWO) [63], Artificial Root Foraging Algorithm (ARFA) [64], Flower Pollination Algorithm (FPA) [65], Phototropic Optimization Algorithm (POA) [66], and Waterwheel Plant Algorithm (WWPA) [67] are included in this subcategory.

Algorithms that model the laws of physics or chemistry fall under the physics/chemistry-based subcategory. Meanwhile, any kind of physical and chemical phenomenon can be a source of inspiration for this subcategory. Physics-based metaheuristic algorithms have gained significant attention in recent research. These algorithms draw inspiration from various physical phenomena to optimize complex problems efficiently. Some of the algorithms in this subcategory are Black Hole Algorithm (BH) [68], Water Wave Optimization (WWA) [69], Lightning Search Algorithm (LSA) [70], Electromagnetic Field Optimization (EFO) [71], Sine Cosine Algorithm (SCA) [72], Thermal Exchange Optimization (TEO) [73], Water Optimization (WAO) [74], Equilibrium Optimizer (EO) [75], Light Spectrum Optimizer (LSO) [76], Prism Refraction Search (PRS) [77], Snow Ablation Optimizer (SAO) [78], and Wave Search Algorithm (WSA) [79].

Certainly, the sky, the galaxy, and the universe are signs of absolute discipline and glory in nature. Sometimes there are unique phenomena in the sky and galaxy that we may negligence many of them. Except for the galaxy we are in (the Milky Way), there are thousands of other galaxies in the universe, each containing stars. So billions of stars are scattered around us [80]. One of the attractions that ordinary people may have paid less attention to is the death of stars. When a star dies, many physical and chemical reactions occur [81]. Studies show that there is some kind of optimization within these reactions. In this paper, we have examined the death of the star from the physical point of view and considered it as the source of inspiration. In this way, in this paper, a novel algorithm called the Star Death (SD) algorithm is introduced and presented. In other words, this paper delves into a metaheuristic algorithm inspired by star death nonlinear physical phenomenon, which presents a solid optimization framework showcasing remarkable exploration and exploitation capabilities for demanding optimization tasks.

The main contribution and innovation of this paper is to consider the physical point of view of the star death process as an effective source of inspiration for creating a new metaheuristic algorithm. By carefully examining this source of inspiration, which is placed under the category of nature-inspired, we find that the wonders of the galaxy happen very precisely, regularly, and optimally. The main goal of this paper is to model the physical process of star

death to present a new lightweight metaheuristic algorithm with a simple mathematical formulation and at the same time more effective compared to other physics-based methods. In addition to having the important and efficient capabilities of nature-inspired algorithms, the proposed new algorithm also introduces new features. The sub-goal of the paper is to perform various experiments to prove the applicability and reliability of the algorithm, especially in the specific application of dynamic load-balancing. Traditional load-balancing methods often rely on static rules or heavy computational models, but the SD algorithm employs an adaptable streamline in fluctuating workloads and a nature-inspired search strategy to rapidly identify near-optimal task allocations. In order to ensure efficient resource utilization and prevent bottlenecks, SD dynamically adjusts to changes in node performance, network latency, or task demands.

Our motivation for formulating an innovative metaheuristic algorithm is derived from several pivotal factors that address the shortcomings of current optimization techniques and the escalating complexity of real-world challenges. The following enumerates our primary motivations:

1. Novel metaheuristic algorithms are inspired by a myriad of sources. This heterogeneity not only cultivates creativity in the algorithmic design but also facilitates investigating distinctive strategies that may culminate in superior optimization outcomes.
2. Our metaheuristic approach is meticulously crafted to enhance adaptability and resilience in addressing various problem types, particularly those that entail constraints. A significant emphasis in the design of our algorithm is the capacity to sustain population diversity and avert premature convergence.
3. There exists a persistent demand for algorithms that exhibit enhanced performance in comparison to their predecessors. Our objective is to augment the efficiency, accuracy, and velocity of optimization processes, which propels the development of our novel algorithm capable of surpassing its forerunners in specific applications.
4. The escalating utilization of metaheuristic methodologies across disciplines such as engineering, artificial intelligence, and complex systems design compels the necessity for perpetual innovation. Our impetus is to develop an algorithm that not only addresses prevailing issues but also exhibits adaptability to emergent challenges in these swiftly advancing areas.
5. As real-world challenges become increasingly intricate, conventional optimization techniques frequently encounter difficulties in yielding satisfactory solutions. Our algorithm, meticulously crafted to navigate complex and high-dimensional landscapes, has emerged as a formidable alternative owing to its flexibility and capacity to transcend local optima.

The continuous development of innovative metaheuristic algorithms is propelled by the imperative for efficient resolutions to progressively intricate optimization challenges, the shortcomings of established methodologies, and the myriad inspirations that catalyze pioneering strategies within this domain.

The remainder of this paper is structured as follows: Sect. 2 introduces star death ideology. Section 3 describes Star Death (SD) algorithm. Section 4 includes experimental results and discussion. Section 5 provides the statistical analysis. Section 6 represents high-dimensional tests. Section 7 provides comparison results with top CEC 2020 algorithms. Section 8 presents application in solving classical engineering problems. Section 9 proposes a specific application for solving dynamic load-balancing. Finally, Sect. 10 represents conclusions.

2 Star death ideology

Many astronomy enthusiasts have surely heard terms like red giant, white dwarf, neutron star, supernova, etc. These names, which are used to refer to a variety of stars, actually depict different parts of a star's life. Stars are born at some point, and after a lifetime they eventually die. This cycle continues. A cycle that can last between several hundred million years and several billion years. Therefore, millions or billions of years should be spent studying the life cycle of stars from their birth to their death. Since humans do not have much time and the process of changes in a star is very slow compared to the life of a human being, scientists came to a model called the stellar evolution [80] model by studying different stars and examining them. The stellar evolution, which represents the life cycle of a star, is a process that a star goes through during its lifetime and takes different times depending on the mass of the star.

When a star is born, the activities inside the core begin. It is the energy released during activities inside the core that makes the star luminous. During this period, the star slowly feeds on its hydrogen and makes helium through fusion in the core. The energy created from this fusion is transferred through photons. As the hydrogen runs out, the star's life story enters a new phase that depends on the star's initial mass. A path that may lead to a massive explosion or end with the star cooling and fading [81]. The factor that determines the life span of the star and its fate and life path is the initial mass of the star. From the size of a star, its life can be roughly estimated. Smaller stars are younger, and larger stars are nearing the end of their lives.

Larger stars lose their energy quickly due to more activity in the core than smaller stars and therefore have a shorter lifespan.

As mentioned, the path of life and death of a star is determined by its mass. There are two paths for star death, one path is followed by high-mass stars, and the other path by low-mass stars. Although the focus of this study is not high-mass stars, in a brief explanation it can be said that they eventually become a black hole or a neutron star. But low-mass stars include a group of stars whose mass is less than eight times the mass of the Sun. This category of stars includes 95% of all the stars in the universe, and the sun is one of them [82]. Figure 2 shows the life cycle of low-mass stars.

After the low-mass star has burned nearly all of its hydrogen, its core begins to contract and heat up, causing the hydrogen to burn even faster. This created extra energy radiates outward and causes the outer layers of the star to move away from its core. At the same time as the outer layers of the star expand, it cools and as a result, its color becomes redder and redder and the star enters the Red Giant stage [83].

In the red giant stage, the star's core has become hot enough to start burning helium. A type of nuclear fusion process in which heavier helium cores are joined together to form larger cores such as carbon and then oxygen. The burning of helium continues, but eventually, the helium in the core runs out, and after most of the core has been converted into carbon and oxygen atoms, the star has no more fuel to burn. As the last helium particles are burned, the outer layers of the star are separated and spread back into interstellar space. The ejected shell of the star forms a mass called the Planetary Nebula [84].

After the planetary nebula disperses and separates into space, a small, very hot, bare core of the star remains. A mass called the White Dwarf [85]. White dwarfs are the last stage in the life cycle of low-mass stars. White dwarfs have low-energy light photons and also have Earth-sized volumes. But they are very dense and their mass is about the mass of the sun. When the star turns into a white dwarf, the end of the star's life comes. Some scientists believe that the white dwarf will eventually lose its light and become a Black Dwarf, which is practically invisible in space [82].

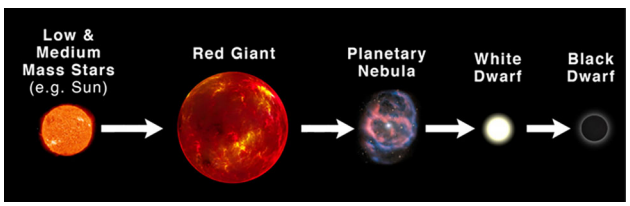

Fig. 2 The life cycle of low-mass stars

The star death ideology is the observation and thinking about physical events and the star death moment, where photons are optimally able to create, carry, or emit energy, and as a result, change the luminosity level of the star.

3 Star Death (SD) algorithm

In a star, all the energy in its center (its core) is made by a process called nuclear fusion. As a result, the energy in the star is released in the form of heat and light. The energy of this heat and light is transferred by photons. A photon, free in a vacuum, is described as a self-sustaining, spiraling wave packet of quantized spin angular momentum, moving at the speed of light [86]. The reason why a star's luminance is this process of moving photons.

In the star death algorithm, it is assumed that the photons are scattered in the body of the red giant. Among these photons, the best photons are the bright photons that are moving on the surface of the red giant, called elite photons. Also, some photons are placed in the center and near the center of the red giant, which are called central photons. The main feature of elite photons is their luminosity and emitting power. The main feature of central photons is their density and nuclear fusion. As we move toward the center of the red giant, the density of photons increases, and as we move toward the periphery of the red giant, the brightness of the photons becomes more apparent. The elite photon and the central photon show the search radius or the radius of the red giant. Over time, the photons are concentrated towards the center and the radius of the red giant gradually decreases until the red giant eventually becomes a white dwarf.

In this algorithm, photons are scattered first. Then, after determining the elite photon, the central photon, the position of these two photons is determined based on the position of other photons in successive iterations. To move the elite photon, first, the luminosity rate is determined through the following equation,

$$L_{rate} = 2.0 \times \alpha \times rand \tag{1}$$

where α is the absorption rate, which is a determinable parameter. This parameter has a direct impact on the process of turning a red giant into a white dwarf and controls the amount of exploration and exploitation. Now that the luminosity rate is obtained, the luminosity level is calculated. We have,

$$L = L_{rate} \times E - p \tag{2}$$

where E is the position of the elite photon and p is the position of the photon. Then the emitter rate is calculated,

$$Em_{rate} = \alpha \times rand - 1.0 \quad (3)$$

where α is the absorption rate parameter. After determining the photon emitter rate, the emitter level is calculated through the following equation,

$$Em = A \times Em_{rate} \times L \quad (4)$$

where A is an adaptive parameter and is obtained through the equation $e^{\left(-2\times\frac{it}{Max_{it}}\right)^2}$. Finally, the new position of the elite photon is obtained through the following relationship,

$$New_E = E - A \times Em \quad (5)$$

The same procedure happens with some changes for the displacement of the central photon. To move the central photon, the density rate must be obtained first. Its equation is very similar to the luminosity rate equation, so we have,

$$D_{rate} = 2.0 \times \alpha \times rand \quad (6)$$

where α is the absorption rate and the rand function is called again to generate a new random number compared to the random number in Eq. (1).

The outward force produced by the fusion process is balanced by the inward gravitational pull of the star. It is this balance between the two that prevents the star from collapsing or expanding. In fact, these photons are constantly approaching each other and causing the density of the star to increase. When the density gets too high, explosions occur and cause the photons to move slightly apart. In this way, the density is slightly reduced. The photon density is calculated based on the density rate of each photon, the central photon and the position of the surrounding photons. Therefore, to calculate the photon density, we have,

$$D = D_{rate} \times C - p \quad (7)$$

where C is the position of the central photon and p is the position of the investigated photon. Now we need to determine the fusion rate, we have,

$$F_{rate} = \alpha \times rand - 1.0 \quad (8)$$

where α is the absorption rate and in general this equation is like the emitter rate equation namely Eq. (3). This process is constantly repeated to maintain the balance of the star. Figure 3 shows the nuclear fusion process which is the source of inspiration for this algorithm. By definition, fusion occurs when two atoms are forced to form a heavier atom. This releases a lot of energy. It should be taken into account that fusion occurs only at extreme density, namely in the center of the star. In Fig. 3, for example, the fusion reaction of Deuterium (D) and Tritium (T) produces a Helium nucleus (or alpha particle) and a high-energy

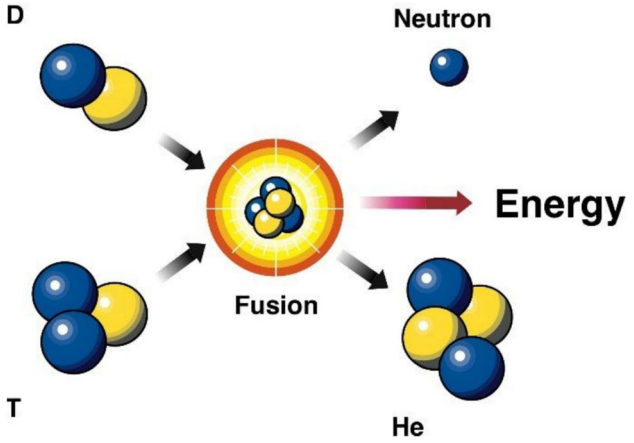

Fig. 3 Example of nuclear fusion reactions

neutron. In this process, there is residual energy, which is the luminosity of the star due to the presence of this energy.

In the SD algorithm, nuclear fusion is calculated through the following equation for the central photon,

$$F = A \times F_{rate} \times D \quad (9)$$

where A is the adaptive parameter that was previously used in Eq. (4). Finally, the new position of the central photon is obtained through the following relationship,

$$New_C = C - A \times F \quad (10)$$

Now, with the obtained data, it is possible to obtain the position of the photon during nuclear fusion. In this way, the position of the photon after fusion and after determining the new position of the elite photon and the central photon will be obtained by averaging the two positions of the elite photon and the central photon. This equation is,

$$p = \frac{New_E + New_C}{2} \quad (11)$$

Figure 4 shows a subjective perception of photon movements. When photons spread around the nucleus, the elite and central photons are identified. The difference between the elite photon and the central photon shows the search radius or the radius of the red giant. Over time, the photons are compressed towards the center and the radius of the red giant gradually decreases until the red giant eventually becomes a white dwarf. This happens through continuous averaging between the central photon and the elite photon, which determines the position of the other photons.

The above conditions are performed for all photons in all their dimensions so that finally the vector $\overrightarrow{p}$ is prepared for the next iteration. In a dying star, this process continues until all the energy of the star's luminosity is lost, thus the star gradually turns from a red giant into a white dwarf. For the proposed algorithm we considered all star interactions

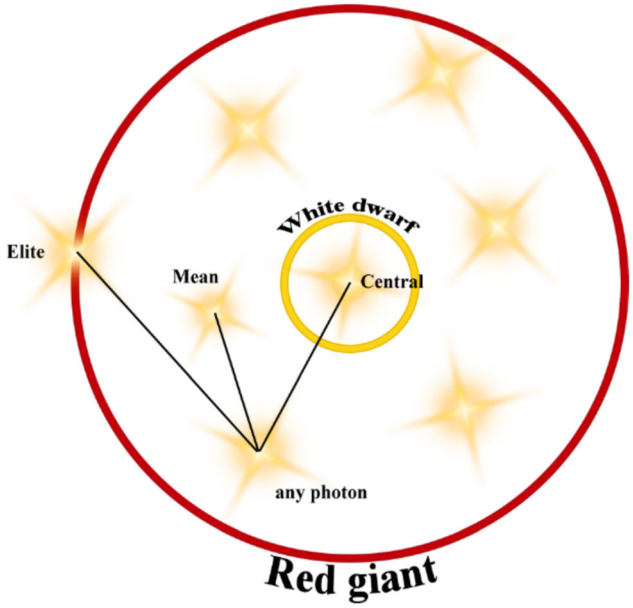

Fig. 4 Subjective perception of photon movements

from a physical point of view. If we were to discuss the matter from a chemical point of view, we should have added pressure equations and equilibrium composition between hydrogen and helium to the discussion, and this would have complicated the algorithm and its calculations. Therefore, the investigation of the star's death from a chemical point of view has not been done. Figure 5 shows the pseudo-code of the Star Death (SD) algorithm. Also, Fig. 6 shows the flowchart of the proposed algorithm.

Fig. 5 Pseudo-code of the Star Death (SD) algorithm

```
STEP 1:
   generate initial swarm array of photons (Swarm size);
   generate position of photons;
   initialize elite and central photons of star;
   STEP 2: for FirstIteration to MaxIteration
      STEP 3: for i=1 to n do (all n photons)
         calculate cost of photon;
         if cost < elite photon
            determine photon as elite photon;
         else
            determine photon as central photon;
         end if
      END STEP 3
      STEP 4: for i=1 to n do (all n photons)
            for j=1 to m do (all m dimensions)
               determine new position of elite photon (Eq. 5);
               determine new position of central photon (Eq. 10);
               determine new position of photon (Eq. 11);
            end for
      END STEP 4
   END STEP 2
END STEP 1
```

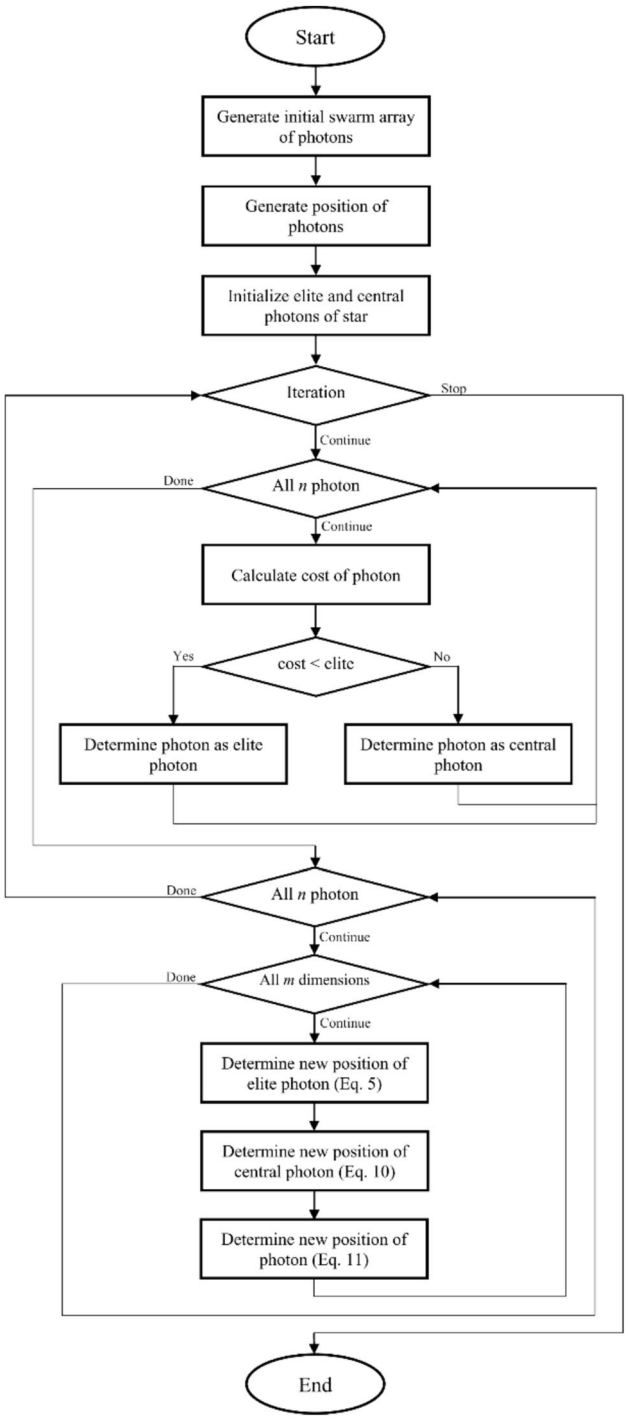

Fig. 6 Flowchart of the SD algorithm

4 Experimental results and discussion

In this section, the experiments performed to evaluate the performance of the proposed SD algorithm are described in detail, and also the discussions are presented about the obtained results. Indeed, we first provide details of the benchmark functions used to test the algorithms. Then, we

explain the parameter settings and conditions for applying the algorithm to the test functions. After that, we present the test results and explain the results. Finally, we analyze the results and identify the characteristics of the SD algorithm.

For performance evaluation, 45 standard benchmark test functions have been used [87]. In applied mathematics, these functions are used to evaluate the properties of optimization algorithms. These functions are designed to check different conditions and situations of algorithms. The purpose of using these functions is to check the performance of the algorithm in terms of accuracy, efficiency, and convergence rate in facing different conditions and situations. In this paper, unimodal and multimodal functions are considered. Unimodal functions have two types. These two types are unimodal benchmark functions with predefined and fixed dimension values and unimodal benchmark functions with d-dimension values. Tables 1 and 2 lists these two categories of unimodal functions.

Multimodal functions are also two types. Multimodal benchmark functions with predefined and fixed dimension values and multimodal benchmark functions with d-dimension values are these two types. Tables 3 and 4 show these two types. It has been tried to be as diverse as possible the benchmark functions so that they can well challenge the proposed algorithm and competing algorithms.

In order to validate and evaluate the performance, the proposed algorithm has been compared with 10 popular algorithms, all of which were implemented by us. These algorithms in alphabetical order are Biogeography Based Optimization (BBO) [57], Differential Evolution (DE) [8], Genetic Algorithm (GA) [6], Golden Jackal Optimization (GJO) [50], Giza Pyramids Construction (GPC) [4], Grey Wolf Optimizer (GWO) [42], Invasive Weed Optimization (IWO) [63], Particle Swarm Optimization (PSO) [23], Teaching–Learning-Based Optimization (TLBO) [58], and White Shark Optimizer (WSO) [51].

Experiments for each algorithm have been conducted under completely fair conditions. Here are some settings for the experiments. The initial population is considered to be 20 for all algorithms. The number of decision variables that determine the dimensions of the problem is set to 30 by default for d-dimensional functions. Each algorithm for each benchmark function is run 30 times independently, then the mean and standard deviation of 30 independent runs are recorded. The parameters related to the algorithms are adjusted through trial and test so that they are in their best state to be applied to the benchmark functions. The type of crossover used in the DE algorithm is binomial crossover. On the other hand, the GA algorithm uses arithmetic crossover. The values of all parameters for each algorithm are shown in Table 5. The criteria for stopping algorithms is the number of function evaluations (NFE). The computational complexity of all competing algorithms is similar to each other so that with the number of populations considered and also the number of iterations that is 500, the number of calls of the benchmark function is equal to 10,000 NFEs.

Tables 6 and 7 show the results obtained from applying the proposed algorithm and competing algorithms on the benchmark test functions. Bohachevsky (f_1) and Booth (f_2) functions are both convex, unimodal functions. These two functions are also defined for two-dimensional space. For these two functions, the SD algorithm performs best. GA algorithm recorded the worst performance. For the Easom (f_3) function, the SD algorithm also performs well. This function is a modal ion function defined for two-dimensional space. However, along with SD, other algorithms such as GPC, PSO, and WSO algorithms provided very good performance. The Gramacy & Lee function (f_4) is a simple, and one-dimensional function. This function is also unimodal. For this function, the GPC, TLBO, WSO, and SD performed best.

Matyas function (f_5) is an almost simple, convex, unimodal, differentiable, and non-separable function. This function is defined for two-dimensional space. For this function, only the BBO, GA, and IWO algorithms performed poorly and the rest of the algorithms achieved the desired solution. The Power Sum function (f_6) is unimodal. This function is defined for four-dimensional space. For this function, the best performance is only for the SD algorithm. Also, Schaffer N.2 (f_7) and Schaffer N.4 (f_8) functions are non-convex, unimodal, differentiable, and non-separable functions. These two functions namely Schaffer N.2 and Schaffer N.4 are defined for two-dimensional space. Although the SD algorithm performs well for these two functions, the rest of the algorithms also perform well. The Griewank function (f_9) however covers the d-dimensional space. For this function, the SD algorithm failed to record good performance. For this function, DE, GJO, GPC, GWO, and TLBO algorithms had excellent performance. Also, for this function, the solutions obtained for the IWO algorithm are in the infeasible range.

The three Hyper-Ellipsoid (f_{10}) Perm (f_{11}) and Zakharov (f_{15}) functions are also unimodal. These functions are also defined for d-dimensional space. The best solutions for Hyper-Ellipsoid and Perm are related to the proposed SD algorithm. For these two functions, some algorithms such as IWO and WSO were in the infeasible range. For the Zakharov function, the best solution is related to the SD algorithm, and the GWO is in the infeasible range. The rest of the algorithms gave typical solutions. Functions Sphere (f_{12}), Sum-Powers (f_{13}), and Sum-Squares (f_{14}) are almost simple functions. These functions are also convex, and unimodal. For these three functions, the best performance is related to the SD

Table 1 Unimodal benchmark functions with predefined and fixed dimension value

Function	Equation	Range	Dim	$f(x^*)$
Bohachevsky	$f_1(x) = x_1^2 + 2x_2^2 - 0.3\cos(3\pi x_1) - 0.4\cos(4\pi x_2) + 0.7$	$[-100, 100]$	2	0
Booth	$f_2(x) = (x_1 + 2x_2 + 7)^2 + (2x_1 + x_2 + 5)^2$	$[-10, 10]$	2	0
Easom	$f_3(x) = -\cos(x_1)\cos x_2 \exp(-(x_1 - \pi)^2 - (x_2 - \pi)^2)$	$[-100, 100]$	2	-1
Gramacy & Lee	$f_4(x) = \frac{\sin(10\pi x)}{2x} + (x-1)^4$	$[0.5, 2.5]$	1	-0.8690
Matyas	$f_5(x) = 0.26(x_1^2 + x_2^2) - 0.48x_1x_2$	$[-10, 10]$	2	0
Power Sum	$f_6(x) = \sum_{i=1}^{d}\left[\left(\sum_{j=1}^{d} x_j^i\right) - b_i\right]^2$ where $b = (8,18,44,114)$	$[0, 4]$	4	0
Schaffer N.2	$f_7(x) = 0.5 + \frac{\sin^2(x_1^2 - x_2^2) - 0.5}{[1 + 0.001(x_1^2 + x_2^2)]^2}$	$[-100, 100]$	2	0
Schaffer N.4	$f_8(x) = 0.5 + \frac{\cos(\sin(\lvert x_1^2 - x_2^2\rvert)) - 0.5}{[1 + 0.001(x_1^2 + x_2^2)]^2}$	$[-100, 100]$	2	0.2925

Table 2 Unimodal benchmark functions with d-dimension value

Function	Equation	Range	Dim	$f(x^*)$
Griewank	$f_9(x) = \sum_{i=1}^{d}\frac{x_i^2}{4000} - \prod_{i=1}^{d}\cos\left(\frac{x_i}{\sqrt{i}}\right) + 1$	$[-600, 600]$	d	0
Hyper-Ellipsoid	$f_{10}(x) = \sum_{i=1}^{d}\sum_{j=1}^{i} x_j^2$	$[-65.536, 65.536]$	d	0
Perm	$f_{11}(x) = \sum_{i=1}^{d}\left(\sum_{j=1}^{d}(j+\beta)\left(x_j^i - \frac{1}{j^i}\right)\right)^2$	$[d, d]$	d	0
Sphere	$f_{12}(x) = \sum_{i=1}^{d} x_i^2$	$[-5.12, 5.12]$	d	0
Sum-Powers	$f_{13}(x) = \sum_{i=1}^{d}\lvert x_i\rvert^{i+1}$	$[-1, 1]$	d	0
Sum-Squares	$f_{14}(x) = \sum_{i=1}^{d} ix_i^2$	$[-10, 10]$	d	0
Zakharov	$f_{15}(x) = \sum_{i=1}^{d} x_i^2 + \left(\sum_{i=1}^{d} 0.5ix_i\right)^2 + \left(\sum_{i=1}^{d} 0.5ix_i\right)^4$	$[-5, 10]$	d	0

algorithm. Although GPC and TLBO algorithms also showed acceptable performance for these three functions. Beale functions (f_{16}), Branin function (f_{17}), and Bukin function (f_{18}) are all multimodal and are defined for two-dimensional space. For these three functions, the SD algorithm performs very well. However, IWO and GWO showed poor performances.

Camel Three-Hump function (f_{20}) is a non-convex, multimodal, differentiable, and non-separable function. For this function, all algorithms were able to reach the desired solution. For the Colville function (f_{21}), which is defined for the four-dimensional space, the best solution is related to the SD and DE algorithms. Camel Six-Hump (f_{19}), Cross-In-Tray (f_{22}), Forrester (f_{26}) and Michalewicz (f_{34}) functions are non-convex and non-separable as well as multimodal functions. All algorithms provide suitable solutions for these functions. Although the best solutions are provided by the SD algorithm.

Other functions, such as De Jong (f_{23}), and Drop-Wave (f_{24}), are all multimodal and non-convex. For the De Jong function, the best performance is related to the SD algorithm. However, in the case of the Drop-Wave function, only BBO, GA, PSO, and TLBO algorithms could not reach the desired solutions. The Eggholder function (f_{25}) is a hard function to optimize. The reason is the large numbers it requires for local optima. For this difficult function, the best solution is the SD algorithm. After SD, the GPC algorithm provided the best solution. The rest of the algorithms in this function have a big difference to the desired solution. The three functions Hartmann-3D (f_{27}), Hartmann-4D (f_{28}), and Hartmann-6D (f_{29}) are multimodal and are defined for three, four and six-dimensional space, respectively. For these three functions, the best solution is related to the SD algorithm. However, in the case of the Hartmann-3D function, in addition to SD, DE and WSO algorithms also provided favorable solutions.

Table 3 Multimodal benchmark functions with predefined and fixed dimension value

Function	Equation	Range	Dim	$f(x^*)$
Beale	$f_{16}(x) = (1.5 - x_1 + x_1x_2)^2 + \left(2.25 - x_1 + x_1x_2^2\right)^2 + \left(2.625 - x_1 + x_1x_2^3\right)^2$	$[-4.5, 4.5]$	2	0
Branin	$f_{17}(x) = \left(x_2 - \frac{5.1}{4\pi^2}x_1^2 + \frac{5}{\pi}x_1 - 6\right)^2 + 10\left(1 - \frac{1}{8\pi}\right)\cos x_1 + 10$	$[-5,10]$	2	0.3978
Bukin	$f_{18}(x) = 100\sqrt{\lvert x_2 - 0.01x_1^2\rvert} + 0.01\lvert x_1 + 10\rvert$	$[-3,3]$	2	0
Camel Six-Hump	$f_{19}(x) = \left(4 - 2.1x_1^2 + \frac{x_1^4}{3}\right)x_1^2 + x_1x_2 + \left(-4 + 4x_2^2\right)x_2^2$	$[-3,3]$	2	− 1.0316
Camel Three-Hump	$f_{20}(x) = 2x_1^2 - 1.05x_1^4 + \frac{x_1^6}{6} + x_1x_2 + x_2^2$	$[-5,5]$	2	0
Colville	$f_{21}(x) = 100\left(x_1^2 - x_2\right)^2 + (x_1 - 1)^2 + (x_3 - 1)^2 + 90\left(x_3^2 - x_4\right)^2 + 10.1\left((x_2 - 1)^2 + (x_4 - 1)^2\right) + 19.8(x_2 - 1)(x_4 - 1)$	$[-10,10]$	4	0
Cross-In-Tray	$f_{22}(x) = -0.0001\left(\left\lvert \sin(x_1)\sin(x_2)\exp\left(\left\lvert 100 - \frac{\sqrt{x_1^2+x_2^2}}{\pi}\right\rvert\right)\right\rvert + 1\right)^{0.1}$	$[-10, 10]$	2	− 2.0626
De Jong	$f_{23}(x) = \left(0.002 + \sum_{i=1}^{25}\frac{1}{i+(x_1-a_{1i})^6+(x_2-a_{2i})^6}\right)^{-1}$	$[-65.536,65.536]$	2	0
Drop-wave	$f_{24}(x) = -\frac{1+\cos\left(12\sqrt{x_1^2+x_2^2}\right)}{0.5\left(x_1^2+x_2^2\right)+2}$	$[-5.12, 5.12]$	2	− 1
Eggholder	$f_{25}(x) = -(x_2 + 47)\sin\left(\sqrt{\left\lvert x_2 + \frac{x_1}{2} + 47\right\rvert}\right) - x_1\sin\left(\sqrt{\lvert x_1 - (x_2 + 47)\rvert}\right)$	$[-512, 512]$	2	− 959.6407
Forrester	$f_{26}(x) = (6x - 2)^2\sin(12x - 4)$	$[0,1]$	1	− 6.0207
Hartmann 3D	$f_{27}(x) = -\sum_{i=1}^{4}\alpha_i\exp\left(-\sum_{j=1}^{3}a_{ij}\left(x_j - p_{ij}\right)^2\right)$ where α, a, p are from [87]	$[0,1]$	3	− 3.8628
Hartmann 4D	$f_{28}(x) = \frac{1}{0.839}\left[1.1 - \sum_{i=1}^{4}\alpha_i\exp\left(-\sum_{j=1}^{3}a_{ij}\left(x_j - p_{ij}\right)^2\right)\right]$ where α, a, p are from [87]	$[0,1]$	4	− 3.1345
Hartmann 6D	$f_{29}(x) = -\sum_{i=1}^{4}\alpha_i\exp\left(-\sum_{j=1}^{6}a_{ij}\left(x_j - p_{ij}\right)^2\right)$ where α, a, p are from [87]	$[0,1]$	6	− 3.3223
Holder-Table	$f_{30}(x) = -\left\lvert \sin(x_1)\cos(x_2)\exp\left(\left\lvert 1 - \frac{\sqrt{x_1^2+x_2^2}}{\pi}\right\rvert\right)\right\rvert$	$[-10,10]$	2	− 19.2085
Langermann	$f_{31}(x) = \sum_{i=1}^{5}c_i\exp\left(-\frac{1}{\pi}\sum_{j=1}^{d}\left(x_j - a_{ij}\right)^2\right)\cos\left(\pi\sum_{j=1}^{d}\left(x_j - a_{ij}\right)^2\right)$ where c, a are from [87]	$[0,10]$	2	− 1.4
Levy N.13	$f_{32}(x) = \sin^2(3\pi x_1) + (x_1 - 1)^2\left[1 + \sin^2(3\pi x_2)\right] + (x_2 - 1)^2\left[1 + \sin^2(2\pi x_2)\right]$	$[-10,10]$	2	0
McCormick	$f_{33}(x) = \sin(x_1 + x_2) + (x_1 - x_2)^2 - 1.5x_1 + 2.5x_2 + 1$	$[-3,4]$	2	− 1.9133
Michalewicz	$f_{34}(x) = -\sum_{i=1}^{d}\sin(x_i)\sin^{2m}\left(\frac{ix_i^2}{\pi}\right)$	$[0, \pi]$	2	− 1.8013
Shekel	$f_{35}(x) = -\sum_{i=1}^{10}\left(\sum_{j=1}^{4}\left(x_j - c_{ji}\right)^2 + \beta_i\right)^{-1}$ where β, c are from [87]	$[0,10]$	4	− 10.5364
Shubert	$f_{36}(x) = \left(\sum_{i=1}^{5}i\cos((i+1)x_1 + i)\right)\left(\sum_{i=1}^{5}i\cos((i+1)x_2 + i)\right)$	$[-10,10]$	2	− 186.7309
Trid	$f_{37}(x) = \sum_{i=1}^{d}(x_i - 1)^2 - \sum_{i=2}^{d}x_ix_{i-1}$	$[d^2, d^2]$	6	− 50

Table 4 Multimodal benchmark functions with d-dimension value

Function	Equation	Range	Dim	$f(x^*)$
Ackley	$f_{38}(x) = -20\exp\left(-0.2\sqrt{\frac{1}{d}\sum_{i=1}^{d} x_i^2}\right) - \exp\left(\frac{1}{d}\sum_{i=1}^{d}\cos(2\pi x_i)\right) + 20 + \exp$	$[-32.768, 32.768]$	d	0
Dixon Price	$f_{39}(x) = (x_1 - 1)^2 + \sum_{i=2}^{d} i(2x_i^2 - x_{i-1})^2$	$[-10, 10]$	d	0
Levy	$f_{40}(x) = \sin^2(\pi\omega_1) + \sum_{i=1}^{d-1}(\omega_i - 1)^2\left[1 + 10\sin^2(\pi\omega_i + 1)\right] + (\omega_d - 1)^2\left[1 + \sin^2(2\pi\omega_d)\right]$	$[-10,10]$	d	0
Powell	$f_{41}(x) = \sum_{i=1}^{\frac{d}{4}}\left[(x_{4i-3} + 10x_{4i-2})^2 + 5(x_{4i-1} + x_{4i})^2 + (x_{4i-2} + 2x_{4i-1})^4 + 10(x_{4i-3} + x_{4i})^4\right]$	$[-4,5]$	d	0
Rastrigin	$f_{42}(x) = 10d + \sum_{i=1}^{d}\left[x_i^2 - 10\cos(2\pi x_i)\right]$	$[-5.12, 5.12]$	d	0
Rosenbrock	$f_{43}(x) = \sum_{i=1}^{d-1}\left[100(x_{i+1} - x_i^2)^2 + (x_i - 1)^2\right]$	$[-5, 10]$	d	0
Schwefel	$f_{44}(x) = 418.9829d - \sum_{i=1}^{d} x_i\sin\left(\sqrt{\lvert x_i\rvert}\right)$	$[-500,500]$	d	0
Styblinski Tang	$f_{45}(x) = \frac{1}{2}\sum_{i=1}^{d}\left(x_i^4 - 16x_i^2 + 5x_i\right)$	$[-5,5]$	d	$-39.1659d$

The Holder-Table function (f_{30}) is non-convex, non-differentiable and non-separable. Only DE and GWO algorithms did not provide good solutions for this function. Also, the SD algorithm has provided the best solution among the algorithms. Also, the Langermann function (f_{31}), which is a function defined for two-dimensional space. The best solution is provided by the BBO algorithm. For this function, the SD algorithm has provided the second-best solution. Levy N.13 (f_{32}), and Shubert (f_{36}) functions are non-convex, multimodal, differentiable, and non-separable. Also, these two functions are considered complex functions. For the Levy N.13 function, the best performance is related to the SD algorithm, and the rest of the algorithms could not have an acceptable performance. Also, for the Shubert function, the SD algorithm together with the PSO algorithm recorded the best performance.

Ackley (f_{38}) and Levy (f_{40}) functions are also non-convex, multimodal, differentiable, and non-separable functions. These two functions are also complex. For the Ackley function, the best performance is related to the SD algorithm, and the worst is related to IWO. In the case of Levy, the DE algorithm has shown the best performance. The three functions of Dixon Price (f_{39}), Rastrigin (f_{42}), and Rosenbrock (f_{43}) are very hard and differentiable functions that are defined for the n-dimensional space. For these three functions, the SD algorithm provides optimal solutions. Meanwhile, for the Rosenbrock function, the WSO algorithm is in the infeasible range. The Powell function (f_{41}) is non-convex and multimodal and is defined for d-dimensional space. SD and GPC provided the best performance for this function, respectively. The Schwefel function (f_{44}) is a function for the d-dimensional space. This function has a large search space and therefore creates a great challenge for the algorithm. The best solution for this function is the SD algorithm and then the GPC algorithm. For the remaining functions namely the Shekel function (f_{35}), Trid function (f_{36}), and Styblinski Tang (f_{45}), the performance of the proposed algorithm is generally evaluated well.

In total, the results of the benchmark functions show that the proposed algorithm is successful. The results show that the SD algorithm could not record the best performance in only three of the 45 benchmark functions.

To show how to search and converge, we randomly selected eight benchmark test functions and recorded the perspective view of the population in iterations 1, 5, and 10. This view is shown in Fig. 7.

To compare the convergence of the proposed algorithm compared to other competing algorithms, we randomly selected 12 benchmark test functions and drew the convergence curve of the algorithms. Figures 8 and 9 show this comparison.

Table 5 The values used to adjust the parameters of the algorithms

Algorithm	Parameters	Values
BBO	Population size	20
	Keep rate	0.2
	Number of kept habitats	4
	Number of new habitats	16
	Immigration rates	[0,1]
DE	Population size	20
	Lower bound of scaling factor	0.2
	Upper bound of scaling factor	0.8
	Crossover probability	0.2
GA	Population size	20
	Crossover percentage	0.7
	Mutation percentage	0.3
GJO	Search agents	20
	C_1	1.5
GPC	Population size	20
	Angle of ramp	10
	Initial velocity	[0,1]
	Minimum friction	1
	Maximum friction	5
	Substitution probability	0.9
GWO	Search agents	20
	Initial convergence factor	2
IWO	Population size	20
	Maximum population size	15
	Minimum number of seeds	0
	Maximum number of seeds	5
	Variance reduction exponent	2
	Initial standard deviation	0.5
	Final standard deviation	0.001
PSO	Swarm size	20
	Inertia weight	1
	Inertia weight damping ratio	0.98
	Personal learning coefficient	2
	Global learning coefficient	2
TLBO	Population size	20
WSO	Population size	20
	Maximum wavy motion	0.75
	Minimum wavy motion	0.07
SD	Swarm size	20
	Absorption rate	1

To calculate the computational complexity of the SD algorithm, it is necessary to calculate the computational complexity and space complexity. In the SD algorithm, there is a main loop that counts the number of iterations. This loop itself contains two separate loops. A loop examines all members of the population and identifies the elite photon and the central photon. The other loop checks all the members of the population again and according to the decision variables, obtains the new position relative to the placement location of the elite photon and the central photon, and finally calculates the position of the new photon based on the mean of these two photons. On the other hand, to define the group of photons, we need $O(n \times Max_{it})$, where n is the number of photons or members of the population, and Max_{it} is the number of iterations of the algorithm. Thus, if we assume that dim is the number of decision variables and the algorithm stops at Max_{it}, the computational complexity is equal to, $O(n \times Max_{it}) + O(n \times Max_{it} \times dim)$. Space complexity is the amount of space required at any given moment. This algorithm is memory-less, as a result, the space complexity of the SD algorithm is equal to $O(dim)$ where dim is the number of decision variables or problem dimensions.

Lightweight metaheuristic algorithms have been devised in diverse fields to tackle issues related to computational complexity. Given the constraints of the operational environment for end-users, there is a pressing need for efficient lightweight optimization algorithms to assist in minimizing unnecessary energy usage and cost. The SD algorithm presents a harmonious combination of optimization effectiveness and computational expenditure across a range of domains. The varied applications underscore the importance of lightweight metaheuristic algorithms in dealing with limitations in resources while upholding performance standards. In comparison to existing approaches, the SD algorithm introduced displays quicker convergence and lower memory demands. The memory requirement of the proposed technique is notably smaller in contrast to many conventional background subtraction methods. The SD algorithm illustrates progress in attaining optimal solutions with reduced computational load in diverse optimization scenarios. Furthermore, within the domain of the Internet of Things (IoT), researchers have delved into the utilization of lightweight metaheuristic algorithms to deploy them on resource-constrained IoT devices, aiming to optimize implementation cost and performance. Given the inherent limitations in processing power and memory of embedded smart devices, the development of lightweight algorithms that demand minimal memory for storage is of utmost significance. The space complexity of the SD algorithm is $O(dim)$ and is relatively small when compared to the memory requirements of other algorithms.

The SD algorithm shows exceptional efficiency in maintaining diversity in the population and avoiding falling into the local optimal trap. This algorithm is also very flexible, effective, and stable. This algorithm has a very good capacity to be used in various problems. The strategy of photons in this algorithm is an interesting feature. To effectively facilitate the emergence of photons from a local

Table 6 Results obtained from applying the SD algorithm and competing algorithms on unimodal benchmark test functions

Fn	Algorithm										
	BBO	DE	GA	GJO	GPC	GWO	IWO	PSO	TLBO	WSO	**SD**
f_1	5.6e-09 ± 1.9e-08	**0.0000 ± 0.0000**	0.0005 ± 0.0016	**0.0000 ± 0.0000**	**0.0000 ± 0.0000**	**0.0000 ± 0.0000**	8.5e-08 ± 6.4e-08	**0.0000 ± 0.0000**	**0.0000 ± 0.0000**	**0.0000 ± 0.0000**	**0.0000 ± 0.0000**
f_2	4.9e-07 ± 7.6e-07	2.9735 ± 3.4317	0.0006 ± 0.0008	6.4e-06 ± 6.5e-06	0.0080 ± 0.0073	1.0e-08 ± 1.2e-08	7.1e-09 ± 6.4e-09	**0.0000 ± 0.0000**	**0.0000 ± 0.0000**	1.4e-22 ± 8.e-22	**0.0000 ± 0.0000**
f_3	− 0.9333 ± 0.2537	− 3.1e-05 ± 4.3e-06	− 0.9332 ± 0.2536	− 0.9999 ± 1.2e-05	**-1.0000 ± 0.0000**	− 3.0e-05 ± 1.0e-20	− 0.2000 ± 0.4068	**-1.0000 ± 0.0000**	− 0.9666 ± 0.1825	**-1.0000 ± 0.0000**	**-1.0000 ± 0.0000**
f_4	− 0.7333 ± 0.1636	− 2.4120 ± 0.8552	− 0.8690 ± 2.3e-08	− 0.8690 ± 1.8e-07	**-0.8690 ± 0.0000**	− 2.8739 ± 1.6e-09	− 0.8690 ± 3.5e-09	− 0.7974 ± 0.1288	**-0.8690 ± 0.0000**	**-0.8690 ± 0.0000**	**-0.8690 ± 0.0000**
f_5	3.6e-06 ± 6.9e-06	**0.0000 ± 0.0000**	0.0002 ± 0.0005	**0.0000 ± 0.0000**	**0.0000 ± 0.0000**	**0.0000 ± 0.0000**	3.2e-10 ± 2.9e-10	**0.0000 ± 0.0000**	**0.0000 ± 0.0000**	**0.0000 ± 0.0000**	**0.0000 ± 0.0000**
f_6	0.0670 ± 0.06751	21.0672 ± 24.2664	0.2725 ± 0.6906	0.4448 ± 0.4099	0.0037 ± 0.0054	175.50 ± 237.42	0.0009 ± 0.0017	0.0040 ± 0.0053	0.0049 ± 0.0084	0.0064 ± 0.0102	**0.0003 ± 1.1e-19**
f_7	0.0006 ± 0.0009	**0.0000 ± 0.0000**	0.0006 ± 0.0010	**0.0000 ± 0.0000**	**0.0000 ± 0.0000**	**0.0000 ± 0.0000**	0.0604 ± 0.0930	**0.0000 ± 0.0000**	**0.0000 ± 0.0000**	**0.0000 ± 0.0000**	**0.0000 ± 0.0000**
f_8	**0.5000 ± 9.10e-07**	0.5003 ± 0.0001	**0.5000 ± 1.1e-06**	**0.5000 ± 3.6e-07**	**0.5000 ± 2.3e-06**	0.5005 ± 0.0003	0.5001 ± 4.2e-05	**0.5000 ± 6.0e-08**	**0.5000 ± 2.3e-07**	0.5001 ± 2.7e-06	**0.5000 ± 2.5e-07**
f_9	1.0273 ± 0.0122	**0.0000 ± 0.0000**	1.0364 ± 0.0172	**0.0000 ± 0.0000**	**0.0000 ± 0.0000**	**0.0000 ± 0.0000**	Infeasible	0.0358 ± 0.0380	**0.0000 ± 0.0000**	7.6817 ± 3.5442	0.0143 ± 0.0161
f_{10}	20.0994 ± 4.6858	1.5e-11 ± 7.7e-11	30.5092 ± 16.1182	4.3e-17 ± 1.3e-16	9.1e-14 ± 2.8e-13	1.5e-17 ± 7.0e-17	Infeasible	0.0016 ± 0.0019	0.0036 ± 0.0048	Infeasible	**4.2e-20 ± 3.7e-20**
f_{11}	1128.35 ± 472.14	74.356 ± 315.0074	Infeasible	239.70 ± 301.86	85.2931 ± 78.53	96.1345 ± 204.08	Infeasible	981.15 ± 1597.14	492.39 ± 514.40	Infeasible	**47.978 ± 69.269**
f_{12}	0.0102 ± 0.0027	3.7e-13 ± 1.4e-12	0.0132 ± 0.0057	3.0e-12 ± 7.9e-12	1.1e-16 ± 4.7e-16	9.9e-05 ± 2.7e-05	0.0001 ± 3.2e-05	1.1e-06 ± 4.3e-06	0.0001 ± 0.0023	2.2607 ± 1.2441	**7.6e-26 ± 8.7e-26**
f_{13}	1.6e-14 ± 3.9e-14	**0.0000 ± 0.0000**	8.4e-07 ± 2.6e-06	**0.0000 ± 0.0000**	**0.0000 ± 0.0000**	**0.0000 ± 0.0000**	7.3e-07 ± 4.1e-07	1.1e-23 ± 4.9e-23	**0.0000 ± 0.0000**	6.6e-07 ± 1.6e-06	**0.0000 ± 0.0000**
f_{14}	0.4391 ± 0.1126	0.5000 ± 2.7386	0.7757 ± 0.4122	3.6e-49 ± 8.4e-49	9.0e-25 ± 2.4e-24	2.4e-68 ± 1.3e-67	0.0822 ± 0.0982	0.0002 ± 0.0005	**0.0000 ± 0.0000**	122.47 ± 103.24	**0.0000 ± 0.0000**
f_{15}	6.3615 ± 5.0840	0.88595 ± 1.8803	163.13 ± 48.5561	1.3e-13 ± 5.4e-13	2.1e-24 ± 6.5e-24	Infeasible	2.6394 ± 2.0272	11.2692 ± 6.3053	6.4e-12 ± 1.3e-11	203.39 ± 101.02	**0.0000 ± 0.0000**

optimum, during each iteration, the best photon within the current population is initially identified. Subsequently, an elite strategy is implemented on this selected photon, and the exploration range of this photon is dynamically adapted based on the quest for an improved solution. Consequently, a cluster of elite photons is produced surrounding this exploration range. Ultimately, if the freshly generated elite photons surpass the best photon within the current population, the latter is substituted with the newly created elite photon.

Another feature of the algorithm is the use of the center-based sampling method. The utilization of the center-based sampling method in the context of the SD algorithm applies to various phases of the optimization process. It is worth noting that this technique highlights the tendency for the center point within a search space to possess a higher likelihood of proximity to an undisclosed solution, particularly evident as the dimensionality escalates. Consequently, this principle serves to enhance the efficacy of the optimizer in attaining superior solutions. Empirical findings substantiate the pivotal role played by center-based sampling in augmenting the rate of convergence for optimization and search algorithms addressing problems of heightened dimensionality.

A significant metric in the realm of algorithmic complexity is the number of parameters utilized in an algorithm. Metaheuristics that make use of a large number of parameters demonstrate a higher level of complexity in comparison to techniques that employ a minimal amount of parameters, influenced by multiple factors. The SD algorithm performs the adjustment or comprehension of these parameters via adaptive parameterization. Consequently, this serves as a proactive strategy against intricate parameters, to enhance the clarity and comprehension of the parameter space.

Parameter interactions in the SD Algorithm serve to avert the presence of numerous locally optimal solutions within the parameter space concerning the quality of the solution. From a practical perspective on optimization, the notion of parameter interaction suggests that the effectiveness of optimizing parameters individually is enhanced by the influence of the emitter effect on luminosity and the fusion effect on density. While there is a certain degree of parameter interaction in algorithms that possess a limited number of parameters, it is noteworthy that this interaction intensifies significantly as the number of parameters involved grows. One of these parameter interactions in the SD algorithm is to adjust the amount of exploration and exploitation. In some problems, it is necessary to carry out more exploration to perform better exploitation. This is especially necessary for high-dimensional and difficult problems. In the SD algorithm, the absorption rate parameter does this. We present a simple example in

Table 7 Results obtained from applying the SD algorithm and competing algorithms on multimodal benchmark test functions

Fn	Algorithm				
	BBO	DE	GA	GJO	GPC
f_{16}	0.1804 ± 0.3709	0.5235 ± 0.9378	0.0702 ± 0.2544	0.0508 ± 0.1933	0.0127 ± 0.0288
f_{17}	**0.3978 ± 0.0000**	2.1671 ± 1.0604	0.3979 ± 2.9e-05	0.6374 ± 0.7301	**0.3978 ± 0.0000**
f_{18}	0.1082 ± 0.0536	0.1017 ± 0.0305	0.1018 ± 0.01105	0.1108 ± 0.0198	0.1011 ± 0.0007
f_{19}	**-1.0316 ± 0.0000**	− 1.0047 ± 0.0098	− 1.0316 ± 1.3e-06	− 1.0316 ± 2.4e-07	− 1.0316 ± 1.2e-09
f_{20}	**0.0000 ± 0.0000**	**0.0000 ± 0.0000**	**0.0000 ± 0.0000**	**0.0000 ± 0.0000**	**0.0000 ± 0.0000**
f_{21}	2.7691 ± 2.5424	**0.0000 ± 0.0000**	6.2015 ± 9.4508	1.3324 ± 0.4131	0.0088 ± 0.0378
f_{22}	**-2.0626 ± 0.0000**	− 2.0454 ± 0.0068	− 2.0626 ± 5.3e-07	− 2.0626 ± 1.0e-05	**-2.0626 ± 0.0000**
f_{23}	7.1687 ± 5.4325	8.5745 ± 4.2969	1.9908 ± 0.1005	7.8948 ± 4.7273	1.0311 ± 0.1814
f_{24}	− 0.9360 ± 0.0576	**-1.0000 ± 0.0000**	− 0.9574 ± 0.0305	**-1.0000 ± 0.0000**	**-1.0000 ± 0.0000**
f_{25}	− 799.36 ± 145.37	− 548.65 ± 117.28	− 837.16 ± 100.57	− 877.87 ± 110.94	− 956.15 ± 5.48
f_{26}	− 6.0207 ± 7.1e-08	− 6.0207 ± 0.0000	− 6.0207 ± 1.9e-07	− 6.0207 ± 4.1e-08	− 6.0207 ± 0.0000
f_{27}	− 3.8370 ± 0.1211	**-3.8628 ± 0.0000**	− 3.8370 ± 0.1421	− 3.8578 ± 0.0035	− 3.7398 ± 0.2722
f_{28}	− 3.0869 ± 0.0968	− 3.1338 ± 0.0027	− 3.0631 ± 0.1109	− 3.0868 ± 0.1551	− 3.1345 ± 2.3e-06
f_{29}	− 3.0138 ± 0.0311	− 3.0362 ± 0.0165	− 3.0240 ± 0.0286	− 2.9001 ± 0.2085	− 3.3070 ± 0.0340
f_{30}	− 18.5616 ± 2.4619	− 10.0792 ± 4.0870	− 19.2083 ± 0.0002	− 19.2079 ± 0.0006	− 19.2083 ± 0.0002
f_{31}	**-3.4196 ± 0.8566**	− 3.7891 ± 0.0845	− 3.6408 ± 0.81897	− 3.8712 ± 0.2233	− 3.5483 ± 0.5533
f_{32}	2.4e-09 ± 1.1e-08	1.3e-31 ± 6.6e-47	1.4e-05 ± 3.5e-05	7.9e-06 ± 6.6e-06	1.3e-09 ± 6.6e-09
f_{33}	− 1.9098 ± 0.0187	− 1.5062 ± 0.2796	− 1.9132 ± 1.8e-06	− 1.9132 ± 7.4e-07	− 1.9064 ± 0.0259
f_{34}	− 1.8013 ± 7.5e-13	− 1.0312 ± 0.2429	− 1.8013 ± 3.5e-06	− 1.7745 ± 0.1462	− 1.8013 ± 3.5e-06
f_{35}	− 5.3339 ± 3.5423	− 5.1285 ± 2.71e-15	− 5.3432 ± 3.2301	− 9.8609 ± 2.0366	− 7.2997 ± 3.8474
f_{36}	− 186.73 ± 1.94e-09	− 69.219 ± 39.0103	− 186.73 ± 0.0006	− 186.62 ± 0.4417	− 186.73 ± 5.2e-06
f_{37}	− 49.9986 ± 0.0012	− 5.1066 ± 0.23002	− 49.1604 ± 0.7170	− 44.6449 ± 11.9814	**-50.0000 ± 0.000**
f_{38}	0.7969 ± 0.1225	0.0478 ± 0.18194	1.088 ± 0.3768	4.5e-11 ± 1.2e-11	2.6e-11 ± 4.2e-11
f_{39}	2.7618 ± 1.3480	1.1137 ± 1.0451	11.3610 ± 5.1380	0.6666 ± 6.2e-07	0.6669 ± 0.0004
f_{40}	10.1252 ± 5.9736	**0.0030 ± 0.0165**	0.1225 ± 0.5144	2.0000 ± 0.2031	2.4713 ± 0.0477
f_{41}	0.2826 ± 0.1078	0.0017 ± 0.0021	2.5591 ± 1.8387	2.3e-07 ± 1.0e-06	1.0e-20 ± 2.9e-20
f_{42}	70.2179 ± 19.0315	0.7311 ± 0.8656	7.5476 ± 2.6558	**0.0000 ± 0.0000**	**0.0000 ± 0.0000**
f_{43}	103.5813 ± 56.6767	37.4821 ± 33.5564	124.4132 ± 39.1860	27.9220 ± 0.6682	28.8303 ± 0.1423
f_{44}	5719.20 ± 636.76	9935.58 ± 396.73	5321.67 ± 319.25	8541.25 ± 874.47	5261.36 ± 613.86
f_{45}	− 35.0613 ± 0.9983	− 16.3650 ± 1.7726	− 33.9746 ± 1.1476	− 24.1893 ± 2.7856	− 32.7575 ± 1.3668

Table 7 continued

Fn	Algorithm					
	GWO	IWO	PSO	TLBO	WSO	**SD**
f_{16}	2.3500 ± 1.5864	0.0762 ± 0.2325	0.0508 ± 0.1933	0.0254 ± 0.1391	1.4e-09 ± 4.0e-09	**1.8e-11 ± 1.6e-11**
f_{17}	3.2123 ± 3.4552	**0.3978 ± 0.0000**	**0.3978 ± 0.0000**	**0.3978 ± 0.0000**	0.3979 ± 7.0e-05	**0.3978 ± 0.0000**
f_{18}	0.1011 ± 0.0081	0.2098 ± 0.0610	0.1011 ± 0.0092	0.1054 ± 0.0179	0.1373 ± 0.0319	**0.1010 ± 0.0112**
f_{19}	− 1.0000 ± 2.43e-06	− 1.0316 ± 1.9e-08	**-1.0316 ± 0.0000**	**-1.0316 ± 0.0000**	− 1.0316 ± 2.3e-06	**-1.0316 ± 0.0000**
f_{20}	**0.0000 ± 0.0000**	**0.0000 ± 0.0000**	**0.0000 ± 0.0000**	**0.0000 ± 0.0000**	**0.0000 ± 0.0000**	**0.0000 ± 0.0000**
f_{21}	0.3700 ± 2.0266	0.01393 ± 0.0326	0.0370 ± 0.0540	0.8336 ± 1.8412	0.5728 ± 1.3558	**0.0000 ± 0.0000**
f_{22}	− 2.0362 ± 0.0056	− 2.0511 ± 0.0438	**-2.0626 ± 0.0000**	**-2.0626 ± 0.0000**	− 2.0626 ± 1.4e-05	**-2.0626 ± 0.0000**
f_{23}	12.0830 ± 1.9099	9.7935 ± 7.3064	6.0693 ± 5.2476	1.3599 ± 1.2551	1.4267 ± 1.2896	**0.9980 ± 7.1e-10**
f_{24}	**-1.0000 ± 0.0000**	**-1.0000 ± 0.0000**	− 0.9978 ± 0.0116	− 0.9978 ± 0.0116	**-1.0000 ± 0.0000**	**-1.0000 ± 0.0000**
f_{25}	− 507.40 ± 137.16	− 678.56 ± 145.20	− 730.62 ± 137.26	− 919.70 ± 53.61	− 929.96 ± 63.16	**-957.47 ± 11.87**
f_{26}	− 6.0207 ± 8.9e-09	− 6.0207 ± 1.3e-09	− 6.0207 ± 0.0000	− 6.0207 ± 0.0000	**-6.0207 ± 0.0000**	**-6.0207 ± 0.0000**
f_{27}	− 3.8596 ± 0.0032	− 3.8628 ± 5.0e-07	− 3.7597 ± 0.2672	− 3.7117 ± 0.1627	**-3.8628 ± 0.0000**	**-3.8628 ± 0.0000**
f_{28}	− 3.0995 ± 0.1185	− 3.1345 ± 2.1e-06	− 3.0551 ± 0.1141	− 3.1345 ± 5.8e-09	− 3.1028 ± 0.0823	**-3.1345 ± 0.0000**
f_{29}	− 3.0070 ± 0.0502	− 2.9871 ± 0.0187	− 3.0261 ± 0.0276	− 3.0135 ± 0.0310	− 3.0260 ± 0.0276	**-3.3104 ± 0.0008**
f_{30}	− 10.3457 ± 3.4891	− 16.4544 ± 4.3599	− 17.3199 ± 3.8509	− 19.2084 ± 0.0001	− 19.2084 ± 0.0001	**-19.2085 ± 3.1e-05**
f_{31}	− 3.7671 ± 0.0495	− 4.0184 ± 0.4961	− 3.6294 ± 0.84945	− 4.0647 ± 0.1528	− 4.1492 ± 0.0121	− 3.5675 ± 0.8401
f_{32}	1.3e-09 ± 6.5e-09	4.5e-08 ± 5.1e-08	1.2e-08 ± 6.6e-08	2.2e-08 ± 3.4e-08	2.2e-09 ± 7.3e-09	**4.2e-11 ± 3.2e-11**
f_{33}	− 1.081 ± 0.7204	− 1.9132 ± 4.0e-09	**-1.9132 ± 0.0000**	**-1.9132 ± 0.0000**	**-1.9132 ± 0.0000**	**-1.9132 ± 0.0000**
f_{34}	− 1.0389 ± 0.2587	− 1.8013 ± 9.8e-08	**-1.8013 ± 0.0000**	**-1.8013 ± 0.0000**	**-1.8013 ± 0.0000**	**-1.8013 ± 0.0000**
f_{35}	− 5.1285 ± 2.7e-15	− 4.5249 ± 2.8771	− 6.6696 ± 3.4653	− 7.5983 ± 3.1424	− 9.4187 ± 2.5498	**-10.5364 ± 0.0000**
f_{36}	− 53.491 ± 35.6051	− 186.73 ± 6.8e-06	**-186.73 ± 0.0000**	− 186.73 ± 0.0012	− 185.82 ± 1.0037	**-186.73 ± 0.0000**
f_{37}	Infeasible	**-50.0000 ± 0.000**	**-50.0000 ± 0.000**	− 49.9242 ± 0.1345	− 49.9863 ± 0.0484	**-50.0000 ± 0.000**
f_{38}	0.0822 ± 0.4504	19.0966 ± 0.1785	2.1268 ± 0.6701	2.8e-11 ± 9.9e-11	7.9863 ± 1.7645	**1.4e-12 ± 1.6e-12**
f_{39}	0.6777 ± 0.0608	2.3583 ± 2.9735	1.1357 ± 0.9281	0.6666 ± 2.2e-07	Infeasible	**0.6666 ± 8.1e-06**
f_{40}	2.4332 ± 1.1534	40.1916 ± 9.6260	3.1571 ± 1.9422	2.1467 ± 1.3036	4.3330 ± 2.2662	0.6795 ± 0.0558
f_{41}	1.5e-07 ± 3.4e-07	0.7389 ± 0.4754	0.0109 ± 0.0095	1.5e-10 ± 6.3e-10	33.4558 ± 22.4478	**1.8e-26 ± 2.0e-26**
f_{42}	30.8462 ± 11.0649	119.850 ± 27.4526	53.3632 ± 14.4711	29.5208 ± 24.3641	64.856 ± 19.8101	**0.0000 ± 0.0000**
f_{43}	27.2693 ± 1.9024	48.1854 ± 52.0327	49.0048 ± 31.4726	27.9144 ± 0.3733	Infeasible	**27.1894 ± 0.2549**
f_{44}	10,409.56 ± 478.99	7421.21 ± 582.94	6224.42 ± 866.97	5390.30 ± 748.02	7003.81 ± 1523.55	**5227.50 ± 442.19**
f_{45}	− 14.2250 ± 1.7280	− 32.8356 ± 1.0763	− 33.2443 ± 1.2960	− 31.8800 ± 1.6690	− 30.0110 ± 1.5703	**-38.2115 ± 0.8021**

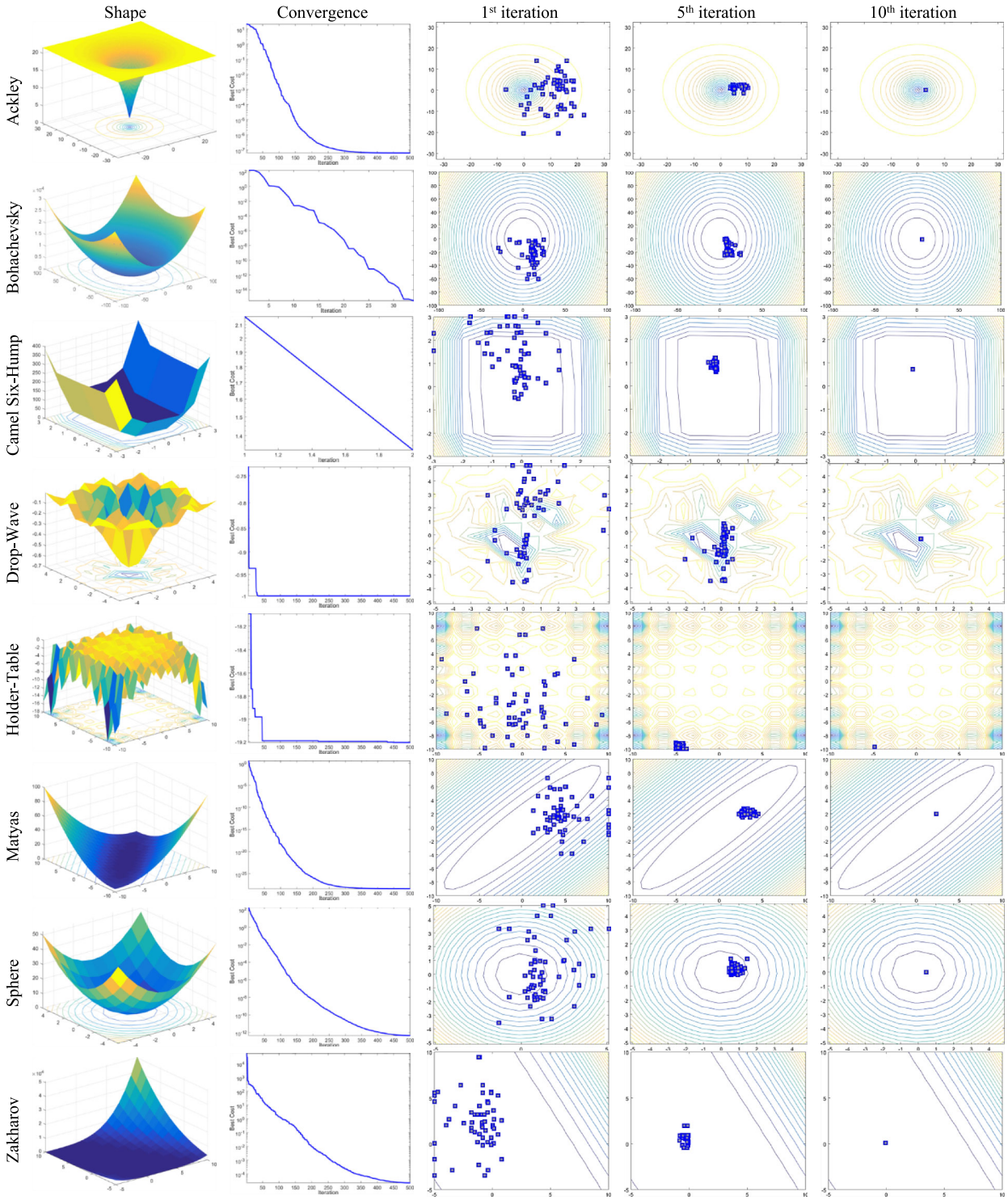

Fig. 7 Function shape, convergence diagram, and perspective view at 1st, 5th, and 10th iteration, respectively, from left to right

Fig. 10. In Fig. 10, the SD algorithm solves the Sphere function as an example. The absorption rate parameter was set once to 1 and again to 4. It can be seen that by setting the parameter to the number 4, the algorithm spends more time in the exploration phase.

5 Statistical analysis

Statistical analysis is the examination of significant differences between algorithms using statistical hypothesis testing. The purpose of statistical analysis is to determine

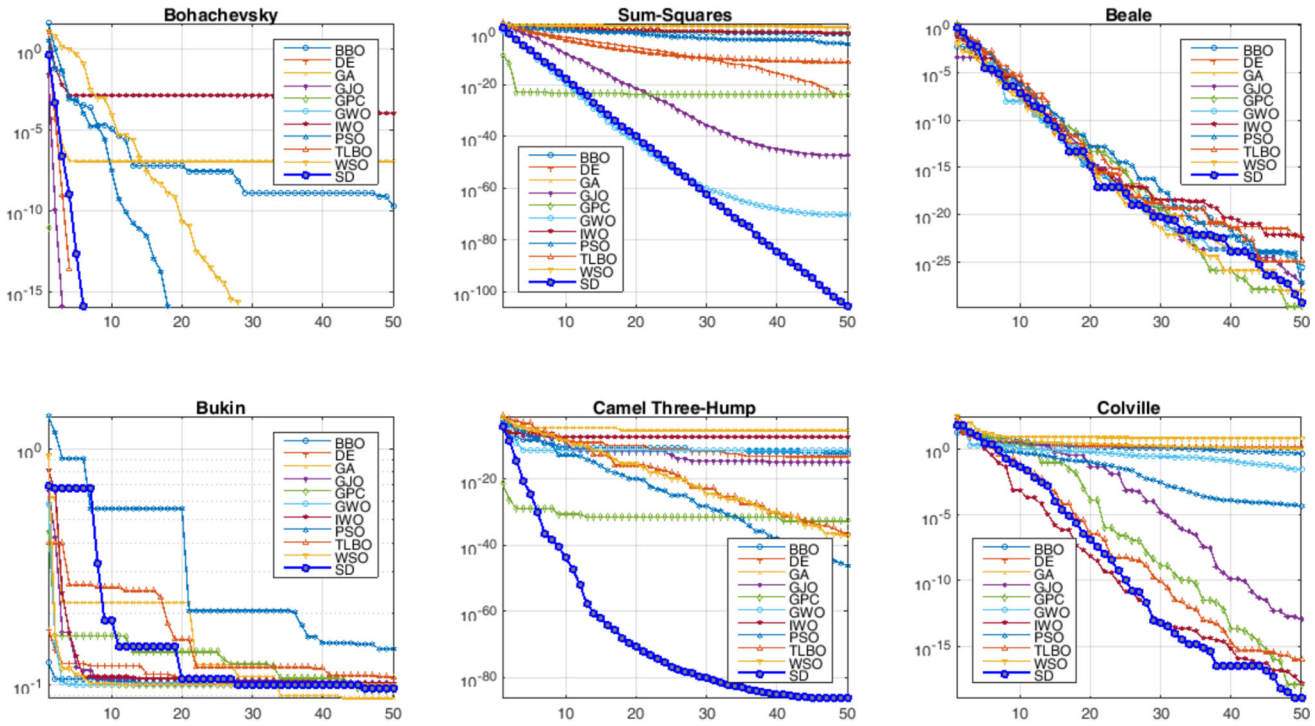

Fig. 8 Convergence curves of algorithms for 12 randomly selected benchmark test functions (Part one)

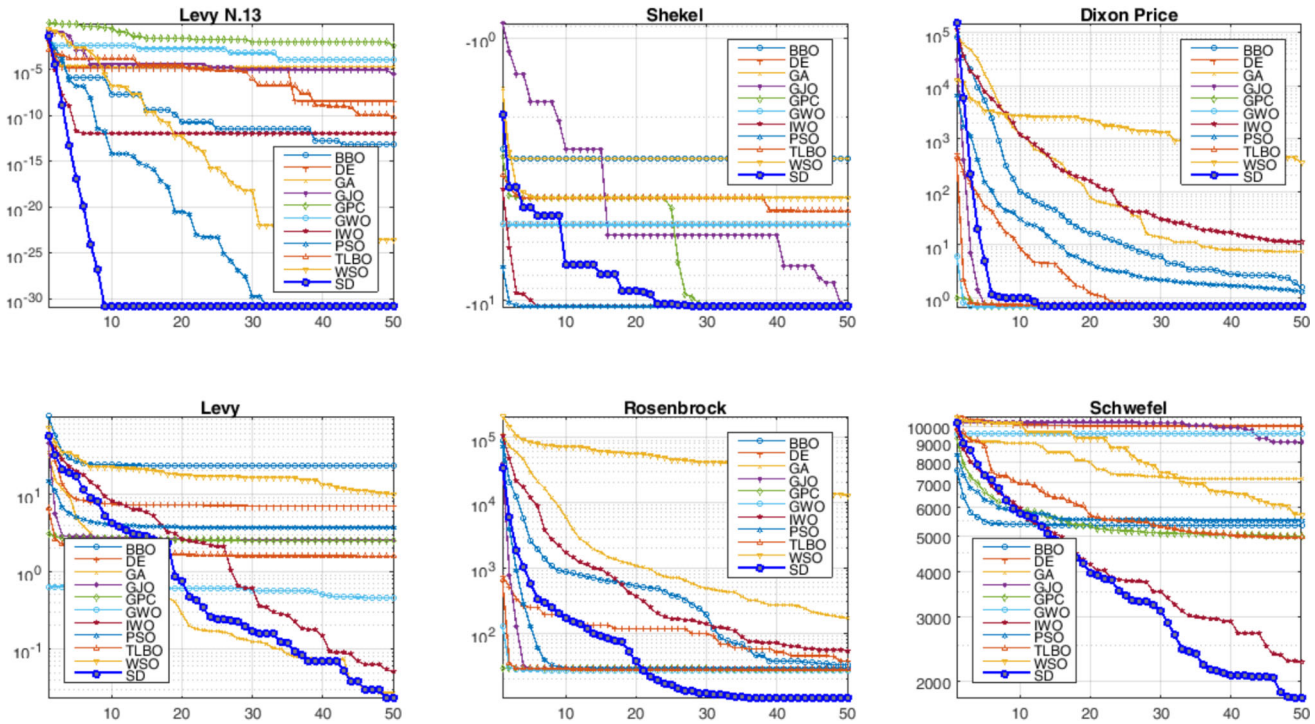

Fig. 9 Convergence curves of algorithms for 12 randomly selected benchmark test functions (Part two)

whether the difference in the results obtained from the algorithms is real or due to a statistical chance of the results obtained. For this purpose, relationships and probabilities between data are determined and investigated quantitatively.

In this paper, Friedman and Iman-Davenport tests are used to find significant differences between the results

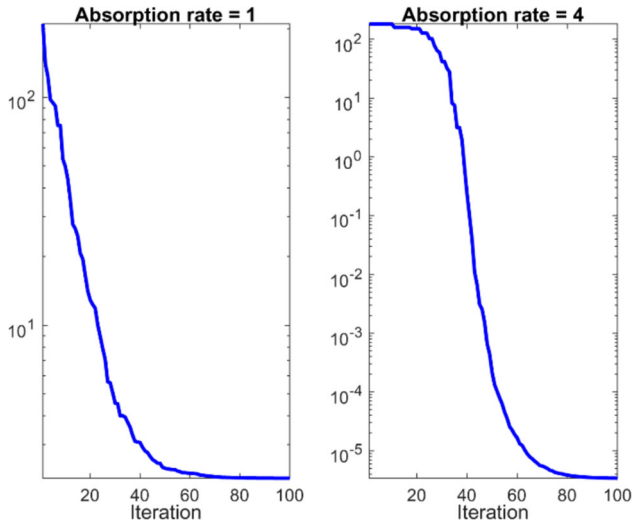

Fig. 10 Adjusting the amount of exploration by changing the absorption rate parameter

Table 8 Ranking of the algorithms (sorted in descending order)

Algorithm	Ranking
SD	2.69
GPC	3.96
TLBO	5.16
GJO	5.76
PSO	5.88
DE	6.52
WSO	6.99
GWO	7.12
BBO	7.13
GA	7.26
IWO	7.54

obtained from the SD algorithm and other competing algorithms. Table 8 shows Friedman's ranking based on the results obtained from two Tables 6 and 7.

As Table 8 shows, according to Friedman's ranking, the SD algorithm has the best ranking, followed by the GPC algorithm and then TLBO.

Table 9 shows the results of Friedman and Iman-Davenport tests. It also shows that the hypothesis is rejected. This table shows that there is a significant difference in the performance of the algorithms.

However, rejecting the hypothesis and identifying the significant level alone is not enough to show a significant difference. To prove the existence of a significant difference, post-hoc tests should be used, which makes a better analysis. In this paper, Holm's method is used as a post-hoc test. In this test, the best rank obtained through the Friedman ranking, which is the SD algorithm, is considered as control algorithm and is compared one by one with other algorithms. Also, consider that the confidence interval for this analysis is 95% ($\alpha = 0.05$). The results obtained from Holm's method are shown in Table 10. The results of the post-hoc test show that the SD algorithm has a significant difference from other competing algorithms.

6 High-dimensional tests

Various sciences and technology are always developing and expanding. This expansion causes the search space to become larger and more complex. Therefore, optimization solutions in high dimensions are felt more than before. Enlarging the search scale is one of the points that we justify in solving optimization problems. Many existing metaheuristics cannot solve high-dimensional problems.

Table 9 Results of Friedman's and Iman–Davenport's tests

Test method	Chi-Square	Degrees of freedom (DF)	*p*-Value	Hypothesis
Friedman	109.3004	10	7.3829e-19	Rejected
Iman–Davenport	12.6450	10	2.2000e-16	Rejected

Table 10 Results of the Holm's method based on the mean of 30 independent runs (SD is the control algorithm)

Algorithm	*j*	α/j	*z*-Score	*p*-Value	Hypothesis
GPC	1	0.05	1.816346233	0.034662	Rejected
TLBO	2	0.025	3.532578893	0.000206	Rejected
GJO	3	0.016666667	4.390695223	< 0.00001	Rejected
PSO	4	0.0125	4.562318490	< 0.00001	Rejected
DE	5	0.01	5.477642575	< 0.00001	Rejected
WSO	6	0.008333333	6.149833701	< 0.00001	Rejected
GWO	7	0.007142857	6.335758906	< 0.00001	Rejected
BBO	8	0.00625	6.350060844	< 0.00001	Rejected
GA	9	0.005555556	6.535986049	< 0.00001	Rejected
IWO	10	0.005	6.936440337	< 0.00001	Rejected

Table 11 The Mean and Standard Deviation of applying the TLBO, GPC and SD algorithms on high-dimensional test functions

Fn	Algorithms					
	TLBO		GPC		SD	
	dim = 5000	dim = 10,000	dim = 5000	dim = 10,000	dim = 5000	dim = 10,000
f_9	1.11e-16 ± 1.12e-16	1.12e-16 ± 1.08e-17	**0.0000 ± 0.0000**	**0.0000 ± 0.0000**	**0.0000 ± 0.0000**	**0.0000 ± 0.0000**
f_{10}	2.91e-16 ± 6.40e-16	1.11e-15 ± 6.14e-15	2.25e-17 ± 2.05e-17	1.74e-16 ± 1.63e-16	**4.12e-20 ± 3.67e-20**	**2.32e-18 ± 2.47e-17**
f_{11}	Infeasible	Infeasible	115.9573 ± 67.8546	191.2547 ± 99.8552	**77.9521 ± 41.6595**	**107.2541 ± 56.7452**
f_{12}	1.16e-15 ± 9.18e-16	1.05e-14 ± 1.17e-14	1.69e-21 ± 1.89e-21	1.38e-22 ± 1.48e-22	**6.99e-34 ± 1.13e-34**	**8.78e-33 ± 1.47e-33**
f_{13}	2.48e-17 ± 2.86e-17	4.14e-16 ± 4.23e-16	3.00e-37 ± 4.68e-37	5.41e-39 ± 7.65e-39	**3.59e-49 ± 2.51e-49**	**2.35e-44 ± 3.14e-44**
f_{14}	6.45e-18 ± 1.38e-19	2.26e-10 ± 2.92e-10	1.36e-18 ± 1.79e-18	9.41e-17 ± 3.50e-17	**8.32e-30 ± 7.02e-32**	**2.92e-28 ± 6.37e-29**
f_{15}	Infeasible	Infeasible	**2.55e-15 ± 4.29e-15**	**3.03e-15 ± 1.59e-15**	7.5005 ± 10.6063	0.0195 ± 0.0266
f_{38}	4.44e-15 ± 4.32e-16	3.14e-15 ± 2.25e-14	4.71e-12 ± 3.97e-12	2.30e-12 ± 2.40 -12	**0.0000 ± 0.0000**	**0.0000 ± 0.0000**
f_{39}	0.9999 ± 5.19e-05	1.0000 ± 1.59e-06	1.0000 ± 1.23e-07	1.0000 ± 3.57e-09	**0.6761 ± 0.0111**	**0.6674 ± 0.0005**
f_{40}	**453.9388 ± 0.6220**	**907.0086 ± 2.1256**	454.2377 ± 0.1675	908.8227 ± 0.0472	454.5420 ± 0.1654	908.8647 ± 0.0091
f_{41}	4.47e-22 ± 6.84e-22	1.12e-24 ± 7.3e-25	6.70e-22 ± 7.18e-22	3.29e-19 ± 4.13e-19	**9.09e-33 ± 3.57e-33**	**1.25e-31 ± 3.32e-32**
f_{42}	**0.0000 ± 0.0000**	**0.0000 ± 0.0000**	**0.0000 ± 0.0000**	**0.0000 ± 0.0000**	**0.0000 ± 0.0000**	**0.0000 ± 0.0000**
f_{43}	4998.841 ± 0.0025	9997.715 ± 0.0883	4998.215 ± 0.0047	9998.893 ± 0.0076	**4997.957 ± 0.0073**	**9996.927 ± 0.0141**
f_{44}	Infeasible	Infeasible	Infeasible	Infeasible	Infeasible	Infeasible
f_{45}	− 14.0846 ± 0.0906	− 14.2084 ± 0.1678	− 9.4854 ± 0.1593	− 9.3132 ± 0.0767	**-15.3816 ± 0.2674**	**-15.5110 ± 0.2019**

Table 12 The results of problems RC01 to RC07 from CEC 2020

Problem	Algorithm			
	SASS	COLSHADE	sCMAgES	SD
RC01	189.3116 ± 5.6843e-14	217.2743 ± 24.9573	189.3116 ± 1.2937e-09	**189.3116 ± 0.0000**
RC02	**7049.037 ± 0.0000**	7049.037 ± 9.0949e-13	7049.037 ± 2.5233e-08	**7049.037 ± 0.0000**
RC03	− 142.7193 ± 2.1540e-05	− 4366.677 ± 318.5757	− 4324.911 ± 263.7816	**-4529.119 ± 0.0007**
RC04	**-0.3882 ± 3.8893e-07**	− 0.3874 ± 0.0028	− 0.3871 ± 0.0036	− 0.3882 ± 2.5612e-05
RC05	**-400.0032 ± 0.0059**	− 340.8304 ± 112.5827	− 397.6837 ± 7.8463	− 399.7448 ± 0.4734
RC06	1.8699 ± 0.0150	2.0633 ± 0.1016	2.0052 ± 0.1293	**1.8638 ± 0.0001**
RC07	1.5739 ± 0.0159	1.8400 ± 0.1912	2.0213 ± 0.1101	**1.5672 ± 0.0002**

The reason is that as the dimensions of the problem increase, its search space increases exponentially. This makes the evaluation of high-dimensional problems very costly. One of the important points is the interaction between variables. If the variables are independent, even if the dimensions are increased, each of the variables can be solved and the whole problem can be solved. But if the variables interact with each other, they all have to be optimized together, so optimization becomes difficult.

In this section, an experiment has been performed to evaluate the performance of the algorithm presented in the high dimensions. The difference between the experiments related to this section and Sect. 4 is the number of dimensions. The number of dimensions is 5000 and 10,000. It should be mentioned that this experiment was performed for benchmark functions for which high dimensions can be considered. This means that experiments have been performed on the benchmark functions of Tables 2 and 4. Also, according to Table 8, the second-best and the third-best algorithms have been used for comparison. According to the table, GPC and TLBO algorithms are selected for comparison. Table 11 shows the results of running the algorithms on the high dimensions. Here we point out that some solutions obtained from some algorithms are not within the acceptable range. Therefore, it is specified in the table with the term infeasible. This means that the algorithm failed to provide a suitable solution.

Table 11 shows that the SD algorithm can solve problems in high dimensions better than the other two algorithms. However, in the case of the Schwefel function (f_{44}),

Table 13 The results of problems RC08 to RC14 from CEC 2020

Problem	Algorithm			
	SASS	COLSHADE	sCMAgES	SD
RC08	**2.0000 ± 0.0000**	**2.0000 ± 0.0000**	2.0000 ± 9.9301e-17	**2.0000 ± 0.0000**
RC09	**2.5576 ± 0.0000**	**2.5576 ± 0.0000**	2.5576 ± 2.7153e-09	2.5579 ± 0.0006
RC10	**1.0765 ± 6.6613e-16**	1.1042 ± 0.0623	1.0765 ± 1.8867e-15	1.0765 ± 4.8000e-05
RC11	101.1912 ± 3.4750	147.8153 ± 20.3821	99.2388 ± 0.0004	**99.2388 ± 0.0003**
RC12	**2.9248 ± 4.4408e-16**	2.9248 ± 4.3546e-16	2.9362 ± 0.0110	2.9276 ± 0.0136
RC13	26,887.42 ± 1.0913e-11	26,887.42 ± 1.4287e-11	26,887.42 ± 1.1082e-11	**26,887.01 ± 0.0004**
RC14	58,505.45 ± 0.0127	58,505.45 ± 2.8574e-11	56,620.40 ± 2967.292	**53,639.04 ± 0.1216**

Table 14 The results of problems RC15 to RC33 from CEC 2020

Problem	Algorithm			
	SASS	COLSHADE	sCMAgES	SD
RC15	2994.424 ± 4.5474e-13	**2994.424 ± 8.9628e-13**	2994.424 ± 2.4556e-12	2994.424 ± 0.0001
RC16	0.0322 ± 1.3877e-17	0.0322 ± 1.3608e-17	0.0364 ± 0.0017	**0.0322 ± 2.0816e-17**
RC17	**0.0126 ± 0.0000**	0.01266 ± 1.0419e-07	0.0126 ± 4.5420e-06	**0.0126 ± 0.0000**
RC18	6059.714 ± 3.6379e-12	6062.179 ± 8.1967	6088.600 ± 65.0244	**5885.361 ± 0.0335**
RC19	1.6702 ± 2.2204e-16	1.6702 ± 2.1773e-16	1.6702 ± 5.2232e-05	**1.6702 ± 6.6613e-16**
RC20	**263.8958 ± 5.6843e-14**	263.8958 ± 5.5739e-14	263.8958 ± 2.1270e-12	263.8969 ± 0.0012
RC21	**0.2352 ± 2.7755e-17**	0.2352 ± 2.7216e-17	0.2352 ± 1.1102e-16	0.2352 ± 1.1102e-16
RC22	1.0015 ± 0.7105	0.5410 ± 0.0417	0.5308 ± 0.0042	**0.5263 ± 0.0004**
RC23	**16.0698 ± 3.5527e-15**	16.0698 ± 3.4837e-15	16.2086 ± 0.2013	**16.0698 ± 3.5527e-15**
RC24	2.5437 ± 0.0000	2.5437 ± 0.0000	2.8297 ± 0.2155	**2.5349 ± 0.0161**
RC25	1616.120 ± 0.0009	1639.037 ± 98.7721	3022.135 ± 387.5627	**1616.120 ± 0.0003**
RC26	38.5140 ± 2.0720	36.6109 ± 1.3411	53.7101 ± 17.5152	**35.3642 ± 0.0060**
RC27	524.4691 ± 0.0064	**524.4507 ± 1.1147e-13**	524.7400 ± 0.1880	524.4518 ± 0.0010
RC28	**14,614.13 ± 0.0000**	16,958.20 ± 7.1346e-12	14,614.13 ± 1.2639e-11	14,614.36 ± 0.2086
RC29	**2,964,895.4 ± 4.65e-10**	2,964,895.4 ± 4.56e-10	2,964,912.3 ± 34.1029	2,964,895.4 ± 0.0346
RC30	2.6585 ± 4.4408e-16	2.6618 ± 0.0108	4.2369 ± 1.0352	**2.6139 ± 4.4408E-16**
RC31	1.81e-18 ± 8.79e-18	1.88e-16 ± 3.73e-16	9.25e-15 ± 2.64e-15	**3.42e-18 ± 1.13e-17**
RC32	− 30,665.53 ± 7.27e-12	− 30,665.53 ± 7.13e-12	− 30,665.53 ± 3.00e-11	**-30,665.48 ± 0.0551**
RC33	2.6393 ± 4.4408e-16	2.6393 ± 4.3546e-16	2.6393 ± 1.6280e-15	**2.6393 ± 8.8817e-16**

none of the algorithms provided a solution within the desired range. But in general, the performance of the SD algorithm is evaluated very well.

7 Analysis of SD on CEC-2020

The CEC 2020 benchmark set is a collection of optimization problems used to evaluate the performance of evolutionary algorithms and other metaheuristics. This set is designed to mimic "general" problems that may be interesting in practice, and it includes a variety of mathematical functions with known properties. These functions are used to test the performance of optimization algorithms in terms of their ability to find good solutions within a given number of function calls. In general, there are 57 real-world optimization problems in this set. The functions in this set are designed to test various aspects of optimization algorithms, such as their ability to handle non-separable functions, non-convex functions, and functions with discontinuities. The

Table 15 The results of problems RC34 to RC44 from CEC 2020

Problem	Algorithm			
	SASS	COLSHADE	sCMAgES	SD
RC34	0.0007 ± 0.0025	4.9548 ± 1.9767	1.2293 ± 0.5623	**0.0003 ± 0.0001**
RC35	0.0803 ± 0.0001	96.0740 ± 20.8837	0.0922 ± 0.0039	**0.0800 ± 5.035e-05**
RC36	0.0479 ± 0.0001	84.3238 ± 19.0493	0.0657 ± 0.0087	**0.0477 ± 1.132e-05**
RC37	0.0189 ± 0.0006	2.6958 ± 0.7765	0.1671 ± 0.1594	**0.0186 ± 4.008e-05**
RC38	2.7378 ± 0.0714	8.2776 ± 1.5964	4.1325 ± 0.8484	**2.7140 ± 9.516e-05**
RC39	3.0095 ± 0.9431	9.3093 ± 2.4904	4.4215 ± 0.7385	**2.7517 ± 0.0001**
RC40	**8.12e-28 ± 1.30e-27**	111.9598 ± 78.3060	1.46e-27 ± 3.51e-28	723.8742 ± 243.7266
RC41	1.45e-26 ± 1.23e-27	18.2764 ± 14.6613	**3.59e-26 ± 8.91e-27**	2.3899 ± 4.7679
RC42	0.0881 ± 0.0082	2.6137 ± 2.1741	8.2803 ± 15.0155	**0.0820 ± 0.0042**
RC43	0.0834 ± 0.0097	24.0294 ± 5.3787	14.6852 ± 18.7252	**0.0830 ± 0.0024**
RC44	− 6109.461 ± 72.4618	− 6032.419 ± 104.1890	− 6085.704 ± 78.5474	**-6135.378 ± 154.5109**

Table 16 The results of problems RC45 to RC50 from CEC 2020

Problem	Algorithm			
	SASS	COLSHADE	sCMAgES	SD
RC45	0.0521 ± 0.0096	**0.0427 ± 0.0054**	0.0856 ± 0.0165	0.0613 ± 0.0212
RC46	0.0542 ± 0.0095	0.0260 ± 0.0055	0.0487 ± 0.0152	**0.0202 ± 6.244e-06**
RC47	0.0462 ± 0.0265	0.0182 ± 0.0031	0.0332 ± 0.0060	**0.0177 ± 0.0123**
RC48	0.0570 ± 0.0192	0.0218 ± 0.0039	0.1332 ± 0.1929	**0.0179 ± 0.0018**
RC49	0.0369 ± 0.0084	0.0325 ± 0.0039	0.1800 ± 0.0881	**0.0129 ± 0.0029**
RC50	0.0236 ± 0.0100	0.0650 ± 0.0472	0.0838 ± 0.0263	**0.0151 ± 0.0003**

Table 17 The results of problems RC51 to RC57 from CEC 2020

Problem	Algorithm			
	SASS	COLSHADE	sCMAgES	SD
RC51	4550.972 ± 0.0586	4550.945 ± 0.0665	4551.003 ± 0.1026	**4550.924 ± 0.1303**
RC52	4165.308 ± 256.6721	3372.124 ± 12.7273	3633.938 ± 54.9228	**3349.019 ± 0.0661**
RC53	5252.365 ± 148.0950	5109.499 ± 55.4132	5466.613 ± 113.3049	**5001.750 ± 4.8793**
RC54	4241.097 ± 2.1175	4245.936 ± 3.3429	4273.1001 ± 8.3910	**4240.842 ± 0.2932**
RC55	6700.402 ± 2.3352	6732.505 ± 53.6035	6727.375 ± 0.2777	**6699.891 ± 4.0427**
RC56	14,751.51 ± 3.6906	14,762.76 ± 4.5762	14,764.76 ± 3.2363	**14,748.93 ± 2.6475**
RC57	3213.308 ± 0.0401	3628.239 ± 287.5174	3312.303 ± 44.2391	**3213.298 ± 0.0067**

classification of these 57 engineering problems is as follows: number of 7 industrial chemical processes (RC01-RC07), number of 7 process synthesis and design problems (RC08-RC14), number of 19 mechanical engineering problems (RC15-RC33), number of 11 power system problems (RC34-RC44), number of 6 power electronic problems (RC45-RC50), and number of 7 livestock feed ration optimization (RC51-RC57). Complete details of these problems are available in [88].

For comparison, the SD algorithm is set based on the conditions in [88]. So that the number of 25 independent implementations is considered. Also, the number of function calls and the number of population are based on the conditions in [88]. A comparison of the mean and standard

deviation of 25 independent runs of the SD algorithm with the top three CEC 2020 algorithms has been done. These algorithms are SASS [89], COLSHADE [90], and sCMA-gES [91], respectively.

Tables 12, 13, 14, 15, 16 and 17 show the results obtained from the SD algorithm compared to the top three CEC 2020 algorithms. As it is clear, the SD algorithm can easily compete with the superior CEC 2020 algorithm. The tables show that this algorithm is successful in solving a large number of these problems.

8 Solving classical engineering problems

Engineering problems usually have several equality and inequality constraints [92]. To solve these problems, a constraint management method is needed. One of the main challenges in constraint management is the direct effects of the fitness function on the position of search agents. In the SD algorithm, there is no direct relationship between the search agents, which are photons, and the objective function. Therefore, it is easy to handle and manage constraints by creating appropriate constraints and penalty functions. In the SD algorithm, if any photon violates the constraint, it is not used in the next iteration. It is necessary to mention that simple and scalar penalty functions have been used in this paper.

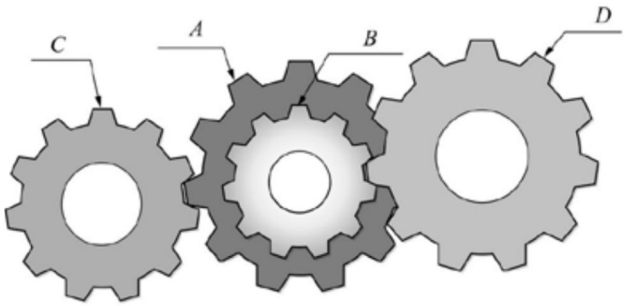

Fig. 11 Parameters and variables of the gear train design problem

Table 18 The results obtained from the algorithms in solving the gear train design problem

Algorithm	Optimum variables				Cost		
	n_A	n_B	n_C	n_D	Best	Mean	Std
BBO	53	13	30	51	2.27e-11	1.83e-07	2.44e-07
DE	53	14	30	50	2.11e-11	1.81e-07	2.76e-07
GA	53	13	30	51	2.30e-11	1.80e-07	3.34e-07
GJO	49	16	19	43	1.38e-12	4.47e-08	4.42e-08
GPC	49	19	16	43	2.70e-12	2.01e-09	2.35e-09
GWO	19	16	44	49	2.78e-11	3.18e-08	3.11e-08
IWO	33	15	13	40	2.15e-08	4.65e-06	3.87e-06
PSO	43	16	19	49	2.70e-12	3.16e-09	5.26e-09
TLBO	43	16	19	49	2.70e-12	3.17e-09	4.94e-09
WSO	33	14	17	50	1.08e-09	2.74e-08	1.16e-07
SDA	49	16	19	43	2.70e-12	1.78e-09	5.05e-09

8.1 Gear train design problem

The gear train design is one of the real-world discrete problems. The goal of this problem is to achieve the optimal tooth value for the four gears of the train so that the gear ratio is minimized. Due to the discreteness of the problem, the search factors are rounded before evaluating the fitness function. Figure 11 shows the parameters and variables of the problem. In this figure, the parameters show the number of teeth of the gears. The number of variables in the problem is four variables. This problem has no constraints. Full details of this problem can be found in Appendix A. Table 18 shows the results obtained from the algorithms.

8.2 Pressure vessel design problem

A pressure vessel is a closed container in which fluids are stored. The design of the vessel is such that the chamber pressure is different from the ambient pressure. Optimum design and construction of these vessels are very important because changes in the pressure difference parameter may cause vessel destruction and explosion. The main objective of this problem is to minimize the cost of materials, forming, and welding of a pressure vessel. Figure 12 shows the parameters and variables of the problem. The variables of this problem are shell thickness (Ts), head thickness (Th), inner radius (R), and cylinder cross-section length excluding the head (L). Details are available in Appendix A. Table 19 shows the results obtained from the algorithms.

8.3 Speed reducer problem

An important part of the gearbox of mechanical systems is the speed reducer. The speed reducer design problem has seven design variables, so the algorithms to solve it face many challenges. The objective is to minimize the weight of the speed reducer in such a way that the constraints are taken into account. The constraints of the problem are the bending stress of the gear teeth, the surface stress, the transverse deviation of the shaft, and the stresses in the shafts. The variables of this problem are the width of the face (b), the module of the teeth (m), the number of teeth in the pinion (z), the length of the first shaft between the

bearings ($l1$), the length of the second shaft between the bearings ($l2$), the diameter of the first shaft ($d1$) and the diameter of the second shaft ($d2$). Figure 13 shows the parameters and variables of the problem. See the details of this problem in Appendix A. Table 20 shows the results obtained from the algorithms.

8.4 Tension/compression spring design problem

The tension/compression spring design problem is a continuous constrained optimization problem. The goal of this problem is to minimize the weight of the tension/compression spring. The variables of the problem are wire diameter (d), average coil diameter (D), and number of active coils (N). Figure 14 shows the parameters and variables of the problem. The conditions and details of this problem can be seen in Appendix A. Table 21 shows the results obtained from the algorithms.

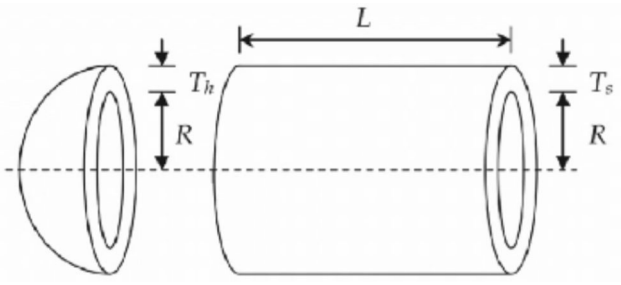

Fig. 12 Parameters and variables of the pressure vessel design problem

8.5 Three-bar truss design problem

A structure consisting of several members, all of which are connected with pins, is called a truss. There is only force in the truss because there are no torque joints in it. The purpose of truss design is to minimize the weight of the truss. For this paper, the design of the three-bar truss is considered. One of the main challenges of this issue is the limited search space and the difficulty of searching in this limited

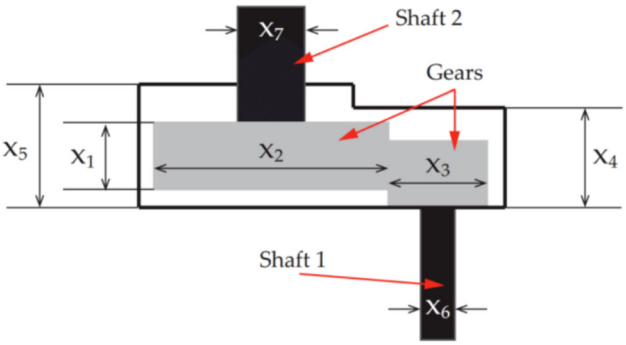

Fig. 13 Parameters and variables of the speed reducer problem

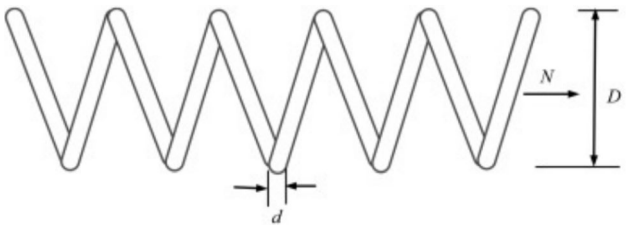

Fig. 14 Parameters and variables of the tension/compression spring design problem

Table 19 The results obtained from the algorithms in solving the pressure vessel design problem

Algorithm	Optimum variables				Cost		
	T_s	T_h	R	L	Best	Mean	Std
BBO	0.8758	0.4330	45.3805	139.8165	6074.7532	6497.4932	342.1000
DE	1.0000	1.0000	48.0610	155.8371	10,215.6706	12,946.8820	773.5101
GA	1.0357	0.5128	52.2626	81.4099	6622.3850	7315.4543	591.3064
GJO	0.7863	0.4153	40.6946	195.4049	5996.9073	6697.4790	533.2893
GPC	0.8090	0.3999	41.9208	178.8500	5940.3298	6196.5115	266.3042
GWO	3.5207	5.9053	84.6510	31.0138	Infeasible	Infeasible	Infeasible
IWO	0.8063	0.3986	41.7704	180.7540	5936.4976	17,635.1938	9259.0283
PSO	0.8669	0.4285	44.9196	144.5550	6055.0596	6241.7073	238.7148
TLBO	0.7781	0.3846	40.3196	199.9999	5885.3352	5995.9059	148.0351
WSO	0.7781	0.3846	40.3199	199.9957	5885.3622	6002.0570	148.9857
SD	0.7789	0.3852	40.3590	199.4544	5887.3137	5887.3137	9.25e-13

Table 20 The results obtained from the algorithms in solving the speed reducer problem

Algorithm	Optimum variables							Cost		
	b	m	z	l_1	l_2	d_1	d_2	Best	Mean	Std
BBO	3.5000	0.7000	17.0000	7.3000	7.7153	3.3502	5.2866	2994.4818	2994.5004	0.0220
DE	3.2727	0.6035	22.9540	8.0571	8.0425	3.5267	5.4830	3544.5543	Infeasible	Infeasible
GA	5.1723	0.7083	12.3816	6.9443	7.7533	3.3565	5.3208	2664.9310	2702.1011	28.5816
GJO	3.5015	0.7000	17.0000	7.3301	8.1154	3.3507	5.2947	3009.4097	3028.5445	8.6294
GPC	3.5000	0.7000	17.0000	7.3000	7.7153	3.3502	5.2866	2994.4719	2994.6103	0.69200
GWO	3.5615	0.7102	24.6215	7.8397	7.9499	3.6914	5.4324	4921.2157	Infeasible	Infeasible
IWO	3.5003	0.7000	17.0000	7.4896	7.7600	3.3513	5.2869	2997.6934	3000.9944	3.1554
PSO	3.5000	0.7000	17.0000	7.3000	7.7153	3.3502	5.2866	2994.4711	2994.4711	1.31e-05
TLBO	3.5000	0.7000	17.0000	7.3000	7.7153	3.3502	5.2866	2994.4711	2996.9166	7.7333
WSO	3.5001	0.7000	17.0001	7.3000	7.7160	3.3502	5.2866	2994.6501	3003.9609	20.423
SD	3.5000	0.7000	17.0000	7.4763	7.7405	3.3544	5.2867	2997.7382	2997.7382	1.38e-12

Table 21 The results obtained from the algorithms in solving tension/compression spring design problem

Algorithm	Optimum variables			Cost		
	d	D	N	Best	Mean	Std
BBO	0.053925	0.41266	8.6376	0.012765	Infeasible	Infeasible
DE	0.053592	0.40311	9.4557	0.013263	0.014330	0.00089
GA	0.055078	0.41964	8.9401	0.013927	Infeasible	Infeasible
GJO	0.050000	0.31709	14.0845	0.012751	0.012860	0.00021
GPC	0.051480	0.35187	11.5784	0.012666	0.012724	7.99e-05
GWO	0.065054	0.66382	8.6975	0.030052	Infeasible	Infeasible
IWO	0.050000	0.31715	14.0671	0.012739	0.012894	0.00014
PSO	0.051968	0.36346	10.9039	0.012667	0.013471	0.00112
TLBO	0.051775	0.35878	11.1695	0.012666	0.012705	4.67e-05
WSO	0.051686	0.35665	11.2929	0.012665	0.012691	5.54e-05
SD	0.050615	0.33142	12.9423	0.012687	0.012687	7.05e-18

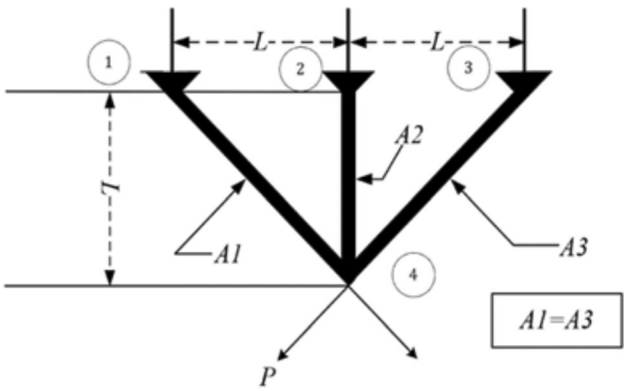

Fig. 15 Parameters and variables of the three-bar truss design problem

space. Constraints of the problem include stress, deflection, and buckling. Figure 15 shows the parameters and variables of the problem. Appendix A describes this problem in detail. Table 22 shows the results obtained from the algorithms.

8.6 Welded beam design problem

The main goal in the problem of welding beam design is to reduce the cost of welding beam fabrication. The minimization of construction cost is influenced by shear stress (τ), bending stress in the beam (θ), buckling load on the bar (Pc), end deflection of the beam (δ), and side constraints. The variables of the problem are weld thickness (h), bar thickness (b), bar height (t), and length of the part attached to the bar (l). Figure 16 shows the parameters and variables of the problem. Details of this problem can be found in Appendix A. Table 23 shows the results obtained from the algorithms.

Table 22 The results obtained from the algorithms in solving three-bar truss design problem

Algorithm	Optimum variables		Cost		
	A_1	A_2	Best	Mean	Std
BBO	0.79137	0.40068	263.9020	266.2802	2.9907
DE	0.78871	0.40814	263.8960	263.8972	0.0023
GA	0.78875	0.40804	263.8967	264.9977	1.4771
GJO	0.78708	0.41284	263.9040	265.8202	5.9811
GPC	0.78875	0.40804	263.8960	263.9000	0.0071
GWO	0.78760	0.41134	263.9018	267.7010	7.9804
IWO	0.78817	0.40967	263.8963	263.8989	0.0031
PSO	0.78877	0.40798	263.8960	263.9027	0.0121
TLBO	0.78877	0.40799	263.8960	263.8960	7.94e-05
WSO	0.78868	0.40825	263.8960	263.8960	5.97e-12
SD	0.78908	0.40711	263.8960	263.8960	1.15e-13

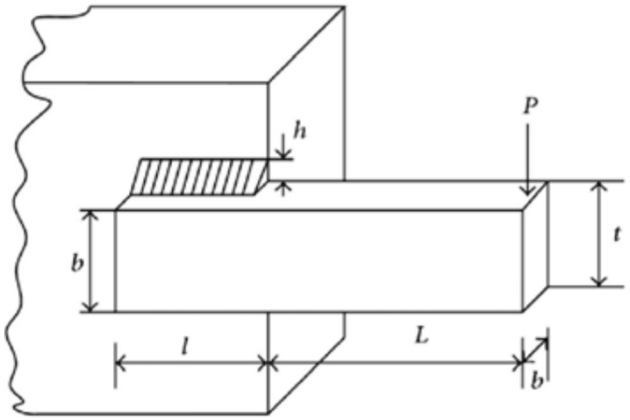

Fig. 16 Parameters and variables of the welded beam design problem

9 Solving dynamic load-balancing as a specific application

Load-balancing is a method of evenly distributing network traffic among the resources that support an application. Modern applications must process millions of users simultaneously and return the correct text, video, images, and other data to each user in a fast and reliable manner. To handle high volumes of traffic, many applications have multiple origin servers with duplicate data. A load balancer is a device that sits between the user and the server group and acts as an invisible facilitator. It also ensures that all source servers are used equally [93]. Dynamic load-balancing is a method used in computing to distribute workloads evenly across a network or computing environment. Unlike static load-balancing, which assigns tasks based on predefined rules, dynamic load-balancing continuously monitors the current load and performance of each server or node in real-time. This allows the system to adaptively allocate incoming tasks based on the current state of resources, maximizing throughput and minimizing response times while avoiding overload on any single resource. Figure 17 provides a visual perception of the load-balancing.

Dynamic load-balancing is essential for maintaining efficient operations in modern computing environments by allowing systems to respond quickly to changing demands. In addition, dynamic load-balancing is particularly beneficial in scenarios with high traffic volumes, complex networks with varying server capacities, and environments where workloads can change suddenly. It is widely used in cluster computing and data centers to ensure optimal performance and resource utilization. Cluster computing

Table 23 The results obtained from the algorithms in solving the welded beam design problem

Algorithm	Optimum variables				Cost		
	h	l	t	b	Best	Mean	Std
BBO	0.28672	2.7052	7.6193	0.28939	2.0178	2.6443	0.33052
DE	0.22223	4.1247	7.7114	0.30368	2.2670	3.0980	0.63903
GA	0.18889	5.4037	6.6958	0.38090	2.5939	3.1556	0.40397
GJO	0.20396	3.5044	9.0529	0.20566	1.7290	1.7361	0.00720
GPC	0.20573	3.4705	9.0366	0.20573	1.7249	1.7710	0.10220
GWO	0.24097	5.2566	8.1159	0.35853	3.0329	Infeasible	Infeasible
IWO	0.19782	3.6581	9.0192	0.20658	1.7409	1.8615	0.13141
PSO	0.20573	3.4705	9.0366	0.20573	1.7249	1.9234	0.36908
TLBO	0.20573	3.4705	9.0366	0.20573	1.7249	1.7249	6.64e-10
WSO	0.20573	3.4705	9.0366	0.20573	1.7249	1.7253	0.00139
SD	0.20571	3.4708	9.0371	0.20573	1.7249	1.7249	6.77e-16

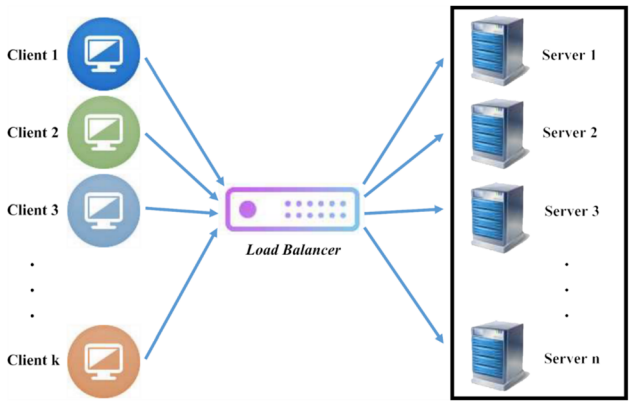

Fig. 17 Visual perception of the load-balancing

represents a paradigm that employs a network of interconnected computing entities (nodes) to function cohesively as a singular system, with the principal objective of augmenting computational capacity and operational efficiency [94]. In contemporary discourse, the incorporation of metaheuristic algorithms within the realm of cluster computing has garnered significant attention, particularly in addressing complex optimization challenges, including those pertinent to clustering endeavors. The application of metaheuristics can be tailored for parallel execution within cluster computing environments [95]. Several metaheuristic algorithms have been proposed to address the complexities of load-balancing in cluster computing environments. These algorithms leverage nature-inspired techniques to optimize resource allocation and improve system performance. PSO [96] is effective in optimizing load distribution by dynamically adjusting the allocation of tasks to nodes based on their performance. ACO [97] excels in finding optimal paths for task allocation, helping to minimize response times and improve resource utilization. GA [98] employs evolutionary principles such as selection, crossover, and mutation to explore the solution space for optimal task distribution, making it suitable for complex load-balancing scenarios. ABC [99] is used for efficient resource allocation by exploring various solutions and selecting the best ones based on their fitness. BA [100] is effective in navigating through solution spaces to achieve balanced loads across resources. WOA [101] focuses on finding optimal solutions through a balance of exploration and exploitation, making it adaptable for dynamic load-balancing. SA [102] can also lead to better overall load distribution. These algorithms have been shown to effectively tackle the challenges associated with load-balancing in cloud computing environments by optimizing performance metrics such as makespan time, response time, and resource utilization.

In this section, a specific application of the proposed SD algorithm in solving the dynamic load-balancing is presented, which is significantly different from other methods in terms of performance. Clusters typically consist of heterogeneous resources with varying capabilities. Designing a metaheuristic algorithm that effectively allocates tasks while considering these differences is complex and requires a nuanced understanding of each resource's performance characteristics. The complexity of load-balancing problems often results in a vast solution space, making it difficult to identify optimal solutions efficiently. This complexity can lead to longer computation times, as the algorithm may struggle to explore all potential allocations effectively. Cluster environments experience fluctuating workloads that can change rapidly. Adapting to these dynamic conditions while maintaining an optimal load distribution is a major challenge, as the system must continuously monitor and adjust allocations in real-time. In cluster computing environments, nodes may fail or become unavailable unexpectedly [103]. Ensuring that SD load-balancing strategies can adapt to such failures without significant performance impacts is crucial for maintaining service continuity. As clusters grow in size and complexity, ensuring that the SD load-balancing method scales effectively becomes increasingly challenging. The algorithm must handle a larger number of nodes and tasks without significant performance degradation. SD algorithm may excel in various performance metrics, such as makespan, response time, and resource utilization. However, optimizing for one metric often adversely affects another, creating trade-offs that must be carefully managed. Integrating the SD algorithm into existing cluster infrastructures can be complex and resource-intensive.

To evaluate the performance in this experiment, the SD algorithm has been compared with the 10 selected algorithms for the competition mentioned in Sect. 4. For this purpose, the number of population is considered 10 for all algorithms. The Number of Function Evaluations (NFE) is also considered to be 1000. We generated random datasets to conduct experiments. So that the load capacity is 1000 per node. Each member of the population represents a node in the network. So there are 10 nodes. Here, the objective function is to calculate the degree of imbalance in the network which through the following equation is obtained,

$$imbalance = |node - average\_load| \tag{12}$$

where the average load is also obtained through,

$$average\_load = \frac{\sum tasks}{nodes} \tag{13}$$

The number of tasks is considered to be 500, 5000, 50,000 and 500,000 respectively. Each algorithm is run 30 times independently and according to the objective function, the maximum average and standard deviation related to the response time and makespan are recorded. Makespan

Table 24 Makespan and response time related to SD and other competing algorithms for the dynamic load-balancing

Algorithm	Criteria	Number of tasks			
		500	5000	50,000	500,000
BBO	Makespan	551.21 ± 13.2562	5517.55 ± 47.6514	56,001.84 ± 814.2513	568,542.54 ± 997.5295
	Response Time	55.3465 ± 2.5654	104.2336 ± 3.2325	104.6874 ± 3.2455	104.5645 ± 3.0254
DE	Makespan	550.14 ± 13.1456	5504.52 ± 49.5514	55,857.67 ± 714.5495	559,756.94 ± 785.5246
	Response Time	55.0015 ± 2.4658	104.1245 ± 3.1246	104.3245 ± 3.0014	104.4614 ± 3.0144
GA	Makespan	550.98 ± 14.6285	5497.52 ± 48.6546	54,987.62 ± 778.2565	550,095.92 ± 875.2548
	Response Time	54.9687 ± 2.1589	104.3254 ± 3.0215	104.6587 ± 3.3325	105.0021 ± 3.6547
GJO	Makespan	540.95 ± 17.2516	5440.17 ± 49.5254	54,361.84 ± 312.5844	534,586.48 ± 586.2145
	Response Time	54.0056 ± 1.9986	100.5589 ± 2.0058	101.0122 ± 1.2564	101.2122 ± 1.3254
GPC	Makespan	526.11 ± 13.0788	5263.26 ± 37.7818	**52,510.27 ± 128.3241**	525,010.38 ± 424.7551
	Response Time	**52.4062 ± 1.4938**	99.9994 ± 0.0098	99.9990 ± 0.0012	99.9946 ± 0.0057
GWO	Makespan	544.69 ± 19.2565	5489.65 ± 55.6598	54,411.77 ± 309.4971	539,961.19 ± 459.6365
	Response Time	54.1458 ± 2.0156	100.9854 ± 2.9548	101.2154 ± 2.7659	101.3554 ± 2.6595
IWO	Makespan	549.48 ± 21.9655	5500.96 ± 54.6593	55,023.95 ± 361.5656	548,755.88 ± 485.6918
	Response Time	54.7565 ± 2.9654	103.2656 ± 3.0254	103.8996 ± 2.6695	103.5887 ± 3.0023
PSO	Makespan	543.95 ± 23.5649	5449.63 ± 51.6695	54,833.84 ± 311.2558	544,339.99 ± 447.6596
	Response Time	53.8993 ± 2.3625	100.5132 ± 2.3659	100.8996 ± 2.8996	101.4256 ± 2.6654
TLBO	Makespan	537.61 ± 18.2568	5332.32 ± 51.2563	54,251.87 ± 211.5362	533,251.24 ± 632.2112
	Response Time	53.8456 ± 1.9865	99.9998 ± 0.0008	99.9999 ± 0.0002	99.9989 ± 0.0125
WSO	Makespan	545.25 ± 19.6698	5511.64 ± 50.2121	54,445.36 ± 308.3368	538,494.33 ± 605.3791
	Response Time	54.9895 ± 2.3659	101.2265 ± 2.6698	102.9856 ± 2.6969	102.7858 ± 2.6524
SD	Makespan	**518.44 ± 15.0279**	**5262.69 ± 36.8927**	**52,510.27 ± 149.7220**	**524,997.88 ± 400.0256**
	Response Time	52.7737 ± 1.7041	**99.9970 ± 0.0013**	**99.4165 ± 0.0014**	**99.4369 ± 0.0013**

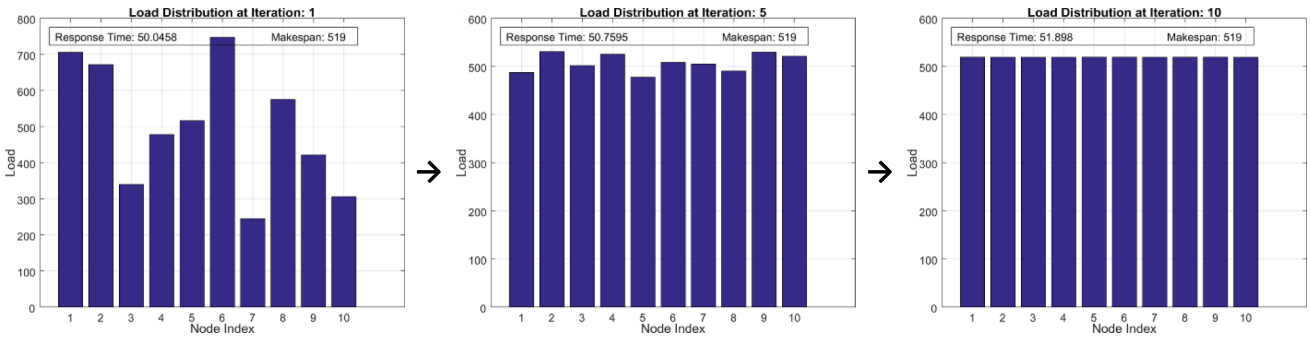

Fig. 18 Performance in load distribution at iteration 1, 5, and 10, respectively (based on one of 30 independent runs)

specifies the completion time of processing of all tasks in the made schedule. The lower this value means, the algorithm can process and deliver jobs faster. Table 24 shows the results of the experiments and comparisons of the SD algorithm with competing algorithms.

As it is clear from the table, the SD algorithm has performed better than other competing algorithms in terms of response time and makespan. This algorithm also performs well in load distribution. Figure 18 shows the load distribution at iteration 1, 5, and 10 of the algorithm. The load distribution is done quickly and this shows the robustness and good efficiency of the SD algorithm.

By considering both task execution time and system resources, SD can significantly reduce load imbalances compared to static methods. SD's inherent mechanism allows it to explore multiple solutions simultaneously, potentially leading to better overall system performance. The algorithm can dynamically adjust to changes in workload, making it effective in environments where demand can fluctuate unpredictably. While SD offers

significant advantages in adaptability and optimization, its complexity may introduce higher computational overhead compared to simpler algorithms like Round Robin. Effective implementation of SD requires careful tuning of parameters and may necessitate more sophisticated monitoring tools. While SD-based load balancing provides enhanced adaptability and optimization for dynamic environments, traditional algorithms like Least Connections offer simplicity and lower overhead but may struggle with uneven workloads. The choice between these approaches depends on specific use cases, system requirements, and performance goals. Figure 19 compares load distribution between the SD algorithm and common load-balancing methods. As the figure shows, the load distribution in the SD algorithm is more balanced than the common methods. Tables 25 and 26 show the comparison of the SD algorithm with common methods and traditional methods, respectively.

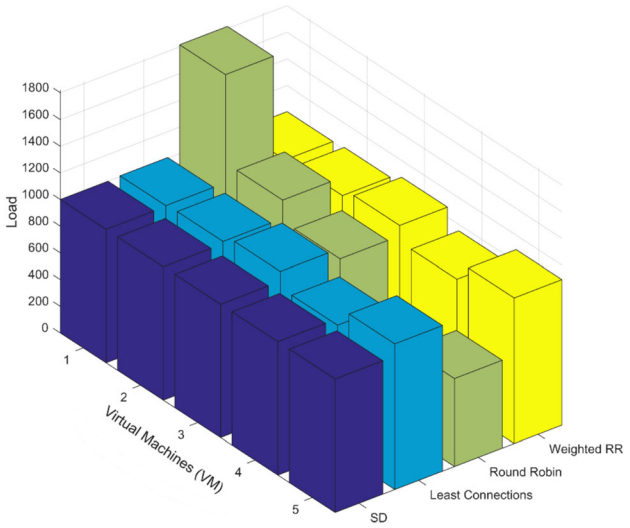

Fig. 19 Comparison of common load-balancing algorithms with SD

The intricacies of load-balancing issues are frequently classified as NP-hard, thereby rendering the identification of optimal resolutions within practical temporal constraints exceedingly challenging. SD offers an approach capable of swiftly generating satisfactory solutions, even in complex contexts. In general, the advantages of SD in solving the load-balancing problem are summarized in the following cases:

1. Adaptability and flexibility: The SD algorithm exhibits a remarkable capacity for adaptation within dynamic environments. This inherent flexibility facilitates the modification of load-balancing strategies in response to fluctuations in workloads and resource availability, a critical requirement in cluster computing contexts where conditions may change rapidly.
2. Exploration and exploitation balance: The SD algorithm proficiently reconciles exploration (the pursuit of new regions within the solution space) and exploitation (the refinement of already established effective solutions). This attitude is essential for the identification of optimal or near-optimal solutions within intricate load-balancing challenges.
3. Domain-agnostic nature: The SD algorithm is broadly applicable across a multitude of domains, extending beyond the realm of computing, thereby establishing it as a versatile tool that can be tailored to various system architectures and characteristics.

Table 25 Comparison SD with common load-balancing algorithms

Feature	Algorithm				
	Round robin	Weighted RR	Least connections	Resource-based	SD load-balancing
Nature	Static	Static with weights	Dynamic	Dynamic	Dynamic and adaptive
Task Distribution	Evenly distributes tasks	Distributes based on assigned weights	Routes to the least busy server	Considers real-time resource usage	Optimizes based on fitness values
Scalability	Simple but can lead to imbalances	Better scalability with weighted nodes	Good scalability, especially under load	Excellent scalability based on metrics	Highly scalable; adapts to workload changes
Complexity	Simple to implement	Moderate complexity due to weight management	Moderate complexity; needs monitoring	High complexity; requires continuous monitoring	More complex due to optimization calculations
Performance	May underperform if nodes are unequal	Better performance with heterogeneous nodes	Efficient under varying connection loads	High performance by optimizing resource use	Can achieve optimal performance through optimization
Overhead	Low overhead	Moderate overhead from weight calculations	Moderate overhead for monitoring connections	High overhead from continuous monitoring	Higher computational overhead due to optimization

Table 26 Comparison SD with traditional dynamic load-balancing algorithms

Feature/Algorithm	SD load-balancing	Traditional dynamic algorithms
Nature	Metaheuristic, adaptive	Heuristic or rule-based
Task distribution	Optimizes based on real-time metrics	Distributes tasks based on predefined rules
Scalability	Highly scalable; adapts to workload changes	Varies; may struggle with high variability
Complexity	More complex due to optimization calculations	Generally simpler and easier to implement
Overhead	Higher computational overhead due to optimization	Lower overhead; often less resource-intensive
Performance	Can achieve optimal performance through optimization	Performance depends on the algorithm used
Energy efficiency	Improved energy consumption through optimized load distribution	May not specifically address energy efficiency

10 Conclusions

In this paper, a novel lightweight algorithm based on physics was proposed called the Star Death (SD) algorithm. The aim was to provide an efficient, lightweight, uncomplicated, simple, and robust algorithm for solving various simple and complex problems. This algorithm demonstrates effectiveness in solving real-world problems, maintaining diversity, and avoiding local optima. Its resilience and effectiveness make it a powerful tool for optimization, especially in engineering challenges. In this algorithm, the search and optimization process is carried out with the inspiration of the process of star death and using two agents, elite photon and central photon. An elite strategy is applied, with its exploration range dynamically adjusted for better solutions. Center-based sampling in the SD algorithm is beneficial throughout the optimization process. The method emphasizes the center point's proximity to solutions, enhancing the optimizer's effectiveness. Empirical evidence supports the role of center-based sampling in improving convergence rates for high-dimensional problems. Parameter interactions in the SD Algorithm help prevent multiple local optima in the parameter space, ensuring solution quality. The SD algorithm adjusts parameters adaptively to enhance clarity and understanding of the parameter space. This algorithm can be used as an effective optimization method in many problems and fields of knowledge, including engineering sciences. Experiments showed that the algorithm can solve problems with high dimensions. The results obtained from the application of the algorithm on real-world problems and classic engineering problems showed that the presented algorithm is fully capable. As a specific application in solving the dynamic load-balancing in cluster computing, it was observed that the SD algorithm is able to deal with the problem. Future research directions for improving metaheuristic load-balancing methods in cluster computing environments can focus on several key areas. Combining different metaheuristic algorithms can enhance performance by leveraging the strengths of each method. Developing algorithms that can dynamically adapt to varying workloads and resource availability is crucial. Future research should explore real-time adjustment mechanisms that allow load-balancing strategies to respond effectively to changes in the cluster environment. Load-balancing often involves optimizing multiple conflicting objectives, such as minimizing response time while maximizing resource utilization. Research can focus on multi-objective optimization methods that effectively balance these trade-offs.

Appendix A

In this appendix, six classic engineering problems are detailed.

Gear train design problem

Consider $\overrightarrow{x} = [x_1 x_2 x_3 x_4] = [n_A n_B n_C n_D]$, the following function should be minimized,

$$f(\vec{x}) = \left(\frac{1}{6.931} - \frac{x_3 x_2}{x_1 x_4}\right)^2 \tag{14}$$

Also, the range of changes of variables are $12 \leq x_1, x_2, x_3, x_4 \leq 60$.

Pressure vessel design problem

Consider $\overrightarrow{x} = [x_1 x_2 x_3 x_4] = [T_s T_h R L]$, the following function should be minimized,

$$f(\vec{x}) = 0.6224 x_1 x_2 x_3 + 1.7781 x_2 x_3^2 + 3.1661 x_1^2 x_4 + 19.84 x_1^2 x_3 \tag{15}$$

subject to,

$$\begin{cases} g_1(\vec{x}) = -x_1 + 0.0193x_3 \le 0 \\ g_2(\vec{x}) = -x_3 + 0.00954x_3 \le 0 \\ g_3(\vec{x}) = -\pi x_3^2 x_4 - \frac{4}{3}\pi x_3^3 + 1296000 \le 0 \\ g_4(\vec{x}) = x_4 - 240 \le 0 \end{cases} \tag{16}$$

Also, the range of changes of variables are $0 \le x_1 \le 99$, $0 \le x_2 \le 99$, $10 \le x_3 \le 200$, and $10 \le x_4 \le 200$.

Speed reducer problem

Consider $\vec{x} = [x_1 x_2 x_3 x_4 x_5 x_6 x_7] = [bmzl_1 l_2 d_1 d_2]$, the following function should be minimized,

$$\begin{aligned} f(\vec{x}) = {} & 0.7854x_1 x_2^2 (3.33x_3^2 + 14.9334x_3 - 43.0934 \\ & - 1508x_1\left(x_6^2 + x_7^2\right) + 7.4777\left(x_6^3 + x_4^3\right) \\ & + 0.7854\left(x_4 x_6^2 + x_5 x_7^2\right) \end{aligned} \tag{17}$$

subject to,

$$\begin{cases} g_1(\vec{x}) = \dfrac{27}{x_1 x_2^2 x_3} - 1 \le 0 \\ g_2(\vec{x}) = \dfrac{397.5}{x_1 x_2^2 x_3^2} - 1 \le 0 \\ g_3(\vec{x}) = \dfrac{1.93x_4^3}{x_2 x_6^4 x_3} - 1 \le 0 \\ g_4(\vec{x}) = \dfrac{1.93x_5^3}{x_2 x_7^4 x_3} - 1 \le 0 \\ g_5(\vec{x}) = \dfrac{\left[\left(745\frac{x_4}{x_2 x_3}\right)^2 + 16.9 \times 10^6\right]^{\frac{1}{2}}}{110x_6^3} - 1 \le 0 \\ g_6(\vec{x}) = \dfrac{\left[\left(745\frac{x_5}{x_2 x_3}\right)^2 + 157.5 \times 10^6\right]^{\frac{1}{2}}}{85x_7^3} - 1 \le 0 \\ g_7(\vec{x}) = \dfrac{x_2 x_3}{40} - 1 \le 0 \\ g_8(\vec{x}) = \dfrac{5x_2}{x_1} - 1 \le 0 \\ g_9(\vec{x}) = \dfrac{x_1}{12x_2} - 1 \le 0 \\ g_{10}(\vec{x}) = \dfrac{1.5x_6 + 1.9}{x_4} - 1 \le 0 \\ g_{11}(\vec{x}) = \dfrac{1.1x_7 + 1.9}{x_5} - 1 \le 0 \end{cases} \tag{18}$$

Also, the range of changes of variables are $2.6 \le x_1 \le 3.6$, $0.7 \le x_2 \le 0.8$, $17 \le x_3 \le 28$, $7.3 \le x_4 \le 8.3$, $7.3 \le x_5 \le 8.3$, $2.9 \le x_6 \le 3.9$, and $5 \le x_7 \le 5.5$.

Tension/compression spring design problem

Consider $\vec{x} = [x_1 x_2 x_3] = [dDN]$, the following function should be minimized,

$$f(\vec{x}) = (x_3 + 2)x_2 x_1^2 \tag{19}$$

subject to,

$$\begin{cases} g_1(\vec{x}) = 1 - \dfrac{x_1^2 x_3}{71785x_1^4} \le 0 \\ g_2(\vec{x}) = \dfrac{4x_2^2 - x_1 x_2}{12566\left(x_2 x_1^3 - x_1^4\right)} + \dfrac{1}{5108x_1^2} \le 0 \\ g_3(\vec{x}) = 1 - \dfrac{140.45x_1}{x_2^2 x_3} \le 0 \\ g_4(\vec{x}) = \dfrac{x_1 + x_2}{1.5} - 1 \le 0 \end{cases} \tag{20}$$

Also, the range of changes of variables are $0.05 \le x_1 \le 2$, $0.25 \le x_2 \le 1.3$, and $2 \le x_3 \le 15$.

Three-bar truss design problem

Consider $\vec{x} = [x_1 x_2] = [A_1 A_2]$, the following function should be minimized,

$$f(\vec{x}) = \left(2\sqrt{2}x_1 + x_2\right) \times l \tag{21}$$

subject to,

$$\begin{cases} g_1(\vec{x}) = \dfrac{\sqrt{2}x_1 + x_2}{\sqrt{2}x_1^2 + 2x_1 x_2} P - \sigma \le 0 \\ g_2(\vec{x}) = \dfrac{x_2}{\sqrt{2}x_1^2 + 2x_1 x_2} P - \sigma \le 0 \\ g_2(\vec{x}) = \dfrac{1}{\sqrt{2}x_2 + x_1} P - \sigma \le 0 \end{cases} \tag{22}$$

where $l = 100cm$, $P = 2kN/cm^2$, and $\sigma = 2kN/cm^2$. Also, the range of changes of variables are $0 \le x_1, x_2 \le 1$.

Welded beam design problem

Consider $\vec{x} = [x_1 x_2 x_3 x_4] = [hltb]$, the following function should be minimized,

$$f(\vec{x}) = 1.10471x_1^2 x_2 + 0.04811x_3 x_4 x_3 (14 + x_2) \tag{23}$$

subject to,

$$\begin{cases} g_1(\vec{x}) = \tau(\vec{x}) - \tau_{max} \le 0 \\ g_2(\vec{x}) = \sigma(\vec{x}) - \sigma_{max} \le 0 \\ g_3(\vec{x}) = \delta(\vec{x}) - \delta_{max} \le 0 \\ g_4(\vec{x}) = x_1 - x_4 \le 0 \\ g_5(\vec{x}) = P - P_c(\vec{x}) \le 0 \\ g_6(\vec{x}) = 0.125 - x_1 \le 0 \\ g_7(\vec{x}) = 1.10471x_1^2 + 0.04811x_3 x_4 (14 + x_2) - 5 \le 0 \end{cases} \tag{24}$$

where $\tau(\vec{x}) = \sqrt{(\tau')^2 + 2\tau'\tau''\frac{x_2}{2R} + (\tau'')^2}$ in such a way that $\tau' = \frac{P}{\sqrt{2}x_1 x_2}$ and $\tau'' = \frac{P\left(L + \frac{x_2}{2}\right)R}{J}$ where in $R = \sqrt{\frac{x_2^2}{4} + \left(\frac{x_1 + x_3}{2}\right)^2}$ and $J = 2\left\{\sqrt{2}x_1 x_2\left[\frac{x_2^2}{4} + \left(\frac{x_1 + x_3}{2}\right)^2\right]\right\}$. Also, we have $\sigma(\vec{x}) =$

$\frac{6PL}{x_4x_3^2}$ and $\delta(\vec{x}) = \frac{6PL^3}{Ex_3^2x_4}$. In addition, we have $P_c(\vec{x}) = \frac{4.013E\sqrt{\frac{x_3^2x^6}{36}}}{L^2}\left(1 - \frac{x_3}{2L}\sqrt{\frac{E}{4G}}\right)$ where $P = 6000lb$, $L = 14in$, $\delta_{max} = 0.25in$, $E = 3 \times 10^6$psi, $G = 12 \times 10^6$ psi, $\tau_{max} = 13600psi$, and $\sigma_{max} = 30000$ psi. Moreover, the range of changes of variables are $0.1 \le x_1 \le 2$, $0.1 \le x_2 \le 10$, $0.1 \le x_3 \le 10$, $0.1 \le x_4 \le 2$.

Author's contributions Sasan Harifi: Conceptualization, Methodology, Resources, Validation, Visualization, Formal analysis, Investigation, Data Curation, Writing—Original Draft, Writing—Review & Editing, Project administration. Reza Eghbali: Conceptualization, Methodology, Software, Writing—Review & Editing. Seyed Mohsen Mirhosseini: Supervision, Writing—Review & Editing, Project administration.

Data availability statement No datasets were generated or analysed during the current study.

Declarations

Conflict of interest The authors declare no competing interests.

References

1. Rao, S.S.: Engineering optimization: theory and practice. Wiley, New York (2019)
2. Hussain, K., Mohd Salleh, M.N., Cheng, S., Shi, Y.: Metaheuristic research: a comprehensive survey. Artif. Intell. Rev. **52**, 2191–2233 (2019)
3. Khanduja, N., Bhushan, B.: Recent advances and application of metaheuristic algorithms: a survey (2014–2020). Metaheur. Evolut. Comput. Algor. Appl., pp 207–228 (2021).
4. Harifi, S., Mohammadzadeh, J., Khalilian, M., Ebrahimnejad, S.: Giza Pyramids Construction: an ancient-inspired metaheuristic algorithm for optimization. Evol. Intel. **14**, 1743–1761 (2021)
5. Slowik, A., Kwasnicka, H.: Evolutionary algorithms and their applications to engineering problems. Neural Comput. Appl. **32**, 12363–12379 (2020)
6. Holland, J.H.: Adaptation in natural and artificial systems: an introductory analysis with applications to biology, control, and artificial intelligence. MIT Press, Cambridge (1992)
7. Neri, F., Cotta, C.: Memetic algorithms and memetic computing optimization: a literature review. Swarm Evol. Comput. **2**, 1–14 (2012)
8. Storn, R., Price, K.: Differential evolution–a simple and efficient heuristic for global optimization over continuous spaces. J. Global Optim. **11**, 341–359 (1997)
9. Geem, Z.W., Kim, J.H., Loganathan, G.V.: A new heuristic optimization algorithm: harmony search. SIMULATION **76**(2), 60–68 (2001)
10. De Castro, L.N., Von Zuben, F.J.: Learning and optimization using the clonal selection principle. IEEE Trans. Evol. Comput. **6**(3), 239–251 (2002)
11. Civicioglu, P.: Backtracking search optimization algorithm for numerical optimization problems. Appl. Math. Comput. **219**(15), 8121–8144 (2013)
12. Salimi, H.: Stochastic fractal search: a powerful metaheuristic algorithm. Knowl.-Based Syst. **75**, 1–18 (2015)
13. Wu, G.: Across neighborhood search for numerical optimization. Inf. Sci. **329**, 597–618 (2016)
14. Kirkpatrick, S., Gelatt, C.D., Jr., Vecchi, M.P.: Optimization by simulated annealing. Science **220**(4598), 671–680 (1983)
15. Glover, F.: Tabu search—part I. ORSA J. Comput. **1**(3), 190–206 (1989)
16. Hansen, P., Mladenović, N., Brimberg, J., Pérez, J.A.M.: Variable neighborhood search, pp. 57–97. Springer, Cham (2019)
17. Balas, E., Vazacopoulos, A.: Guided local search with shifting bottleneck for job shop scheduling. Manage. Sci. **44**(2), 262–275 (1998)
18. Lourenço, H. R., Martin, O. C., Stützle, T.: Iterated local search: Framework and applications. Handbook of Metaheuristics, pp 129–168 (2019).
19. Harifi, S., Khalilian, M., Mohammadzadeh, J., Ebrahimnejad, S.: New generation of metaheuristics by inspiration from ancient. In: 2020 10th international conference on computer and knowledge engineering (ICCKE) (pp. 256–261). IEEE, New York (2020)
20. Guan, Z., Ren, C., Niu, J., Wang, P., Yizi, S.: Great Wall Construction Algorithm: a novel meta-heuristic algorithm for engineer problems. Expert Syst. Appl., 120905 (2023).
21. Niu, J., Ren, C., Guan, Z., Cao, Z.: Dujiangyan irrigation system optimization (DISO): A novel metaheuristic algorithm for dam safety monitoring. In Structures (Vol. 54, pp. 399–419). Elsevier, Amsterdam (2023)
22. Adhikari, M., Srirama, S. N., Amgoth, T.: A comprehensive survey on nature-inspired algorithms and their applications in edge computing: challenges and future directions. Softw. Pract. Experience, **52**(4), 1004–1034 (2022).
23. Kennedy, J., Eberhart, R.: Particle swarm optimization. In: Proceedings of ICNN'95-international conference on neural networks (Vol. 4, pp. 1942–1948). IEEE, New York (1995)
24. Krishnanand, K. N., Ghose, D.: Detection of multiple source locations using a glowworm metaphor with applications to collective robotics. In: Proceedings 2005 IEEE Swarm Intelligence Symposium, 2005. SIS 2005. (pp. 84–91). IEEE, New York (2005)
25. Shah-Hosseini, H.: The intelligent water drops algorithm: a nature-inspired swarm-based optimization algorithm. Int. J. Bio-inspired Comput. **1**(1–2), 71–79 (2009)
26. He, S., Wu, Q.H., Saunders, J.R.: Group search optimizer: an optimization algorithm inspired by animal searching behavior. IEEE Trans. Evol. Comput. **13**(5), 973–990 (2009)
27. Oftadeh, R., Mahjoob, M.J., Shariatpanahi, M.: A novel meta-heuristic optimization algorithm inspired by group hunting of animals: Hunting search. Comput. Math. Appl. **60**(7), 2087–2098 (2010)
28. Duman, E., Uysal, M., Alkaya, A.F.: Migrating birds optimization: a new metaheuristic approach and its performance on quadratic assignment problem. Inf. Sci. **217**, 65–77 (2012)
29. Li, X., Zhang, J., Yin, M.: Animal migration optimization: an optimization algorithm inspired by animal migration behavior. Neural Comput. Appl. **24**, 1867–1877 (2014)
30. Rahmani, R., Yusof, R.: A new simple, fast and efficient algorithm for global optimization over continuous search-space problems: radial movement optimization. Appl. Math. Comput. **248**, 287–300 (2014)
31. Cuevas, E., González, A., Zaldívar, D., Pérez-Cisneros, M.: An optimisation algorithm based on the behaviour of locust swarms. Int. J. Bio-Inspired Comput. **7**(6), 402–407 (2015)
32. Odili, J.B., Kahar, M.N.M., Anwar, S.: African buffalo optimization: a swarm-intelligence technique. Procedia Comput. Sci. **76**, 443–448 (2015)

33. Sun, G., Zhao, R., Lan, Y.: Joint operations algorithm for large-scale global optimization. Appl. Soft Comput. **38**, 1025–1039 (2016)
34. Pierezan, J., Coelho, L. D. S.: Coyote optimization algorithm: a new metaheuristic for global optimization problems. In: 2018 IEEE congress on evolutionary computation (CEC) (pp. 1–8). IEEE, New York (2018).
35. Harifi, S., Khalilian, M., Mohammadzadeh, J., Ebrahimnejad, S.: Emperor Penguins colony: a new metaheuristic algorithm for optimization. Evol. Intel. **12**, 211–226 (2019)
36. Zhang, W., Pan, K., Li, S., Wang, Y.: Special Forces Algorithm: a novel meta-heuristic method for global optimization. Math. Comput. Simul. **213**, 394–417 (2023)
37. Karaboga, D., Basturk, B.: A powerful and efficient algorithm for numerical function optimization: artificial bee colony (ABC) algorithm. J. Global Optim. **39**, 459–471 (2007)
38. Dorigo, M., Birattari, M., Stutzle, T.: Ant colony optimization. IEEE Comput. Intell. Mag. **1**(4), 28–39 (2006)
39. Yang, X. S.: Firefly algorithms for multimodal optimization. In International symposium on stochastic algorithms (pp. 169–178). Springer, Berlin (2009).
40. Yang, X. S.: A new metaheuristic bat-inspired algorithm. In Nature inspired cooperative strategies for optimization (NICSO 2010) (pp. 65–74). Springer, Berlin (2010).
41. Gandomi, A.H., Alavi, A.H.: Krill herd: a new bio-inspired optimization algorithm. Commun. Nonlinear Sci. Numer. Simul. **17**(12), 4831–4845 (2012)
42. Mirjalili, S., Mirjalili, S.M., Lewis, A.: Grey wolf optimizer. Adv. Eng. Softw. **69**, 46–61 (2014)
43. Mirjalili, S.: Moth-flame optimization algorithm: a novel nature-inspired heuristic paradigm. Knowl.-Based Syst. **89**, 228–249 (2015)
44. Mirjalili, S.: The ant lion optimizer. Adv. Eng. Softw. **83**, 80–98 (2015)
45. Askarzadeh, A.: A novel metaheuristic method for solving constrained engineering optimization problems: crow search algorithm. Comput. Struct. **169**, 1–12 (2016)
46. Mirjalili, S., Lewis, A.: The whale optimization algorithm. Adv. Eng. Softw. **95**, 51–67 (2016)
47. Jain, M., Singh, V., Rani, A.: A novel nature-inspired algorithm for optimization: Squirrel search algorithm. Swarm Evol. Comput. **44**, 148–175 (2019)
48. Azizi, M., Talatahari, S., Gandomi, A.H.: Fire Hawk Optimizer: a novel metaheuristic algorithm. Artif. Intell. Rev. **56**(1), 287–363 (2023)
49. Hamad, R. K., Rashid, T. A.: GOOSE algorithm: A powerful optimization tool for real-world engineering challenges and beyond. Evolving Syst., pp 1–26 (2024).
50. Chopra, N., Ansari, M.M.: Golden jackal optimization: a novel nature-inspired optimizer for engineering applications. Expert Syst. Appl. **198**, 116924 (2022)
51. Braik, M., Hammouri, A., Atwan, J., Al-Betar, M.A., Awadallah, M.A.: White Shark Optimizer: a novel bio-inspired meta-heuristic algorithm for global optimization problems. Knowl.-Based Syst. **243**, 108457 (2022)
52. Abdel-Basset, M., Mohamed, R., Jameel, M., Abouhawwash, M.: Spider wasp optimizer: a novel meta-heuristic optimization algorithm. Artif. Intell. Rev. **56**(10), 11675–11738 (2023)
53. Abdollahzadeh, B., Khodadadi, N., Barshandeh, S., Trojovský, P., Gharehchopogh, F. S., El-kenawy, E. S. M., et al.: Puma optimizer (PO): a novel metaheuristic optimization algorithm and its application in machine learning. Cluster Comput., pp 1–49 (2024).
54. Han, M., Du, Z., Yuen, K.F., Zhu, H., Li, Y., Yuan, Q.: Walrus optimizer: A novel nature-inspired metaheuristic algorithm. Expert Syst. Appl. **239**, 122413 (2024)
55. Zervoudakis, K., Tsafarakis, S.: A global optimizer inspired from the survival strategies of flying foxes. Eng. Comput, pp 1–34 (2022).
56. Atashpaz-Gargari, E., Lucas, C.: Imperialist competitive algorithm: an algorithm for optimization inspired by imperialistic competition. In: 2007 IEEE congress on evolutionary computation (pp. 4661–4667). Ieee, New York (2007).
57. Simon, D.: Biogeography-based optimization. IEEE Trans. Evol. Comput. **12**(6), 702–713 (2008)
58. Rao, R.V., Savsani, V.J., Vakharia, D.P.: Teaching–learning-based optimization: a novel method for constrained mechanical design optimization problems. Comput. Aided Des. **43**(3), 303–315 (2011)
59. Elsisi, M.: Future search algorithm for optimization. Evol. Intel. **12**(1), 21–31 (2019)
60. Askari, Q., Younas, I., Saeed, M.: Political optimizer: a novel socio-inspired meta-heuristic for global optimization. Knowl.-Based Syst. **195**, 105709 (2020)
61. Askari, Q., Saeed, M., Younas, I.: Heap-based optimizer inspired by corporate rank hierarchy for global optimization. Expert Syst. Appl. **161**, 113702 (2020)
62. Azizi, M., Baghalzadeh Shishehgarkhaneh, M., Basiri, M., Moehler, R.C.: Squid game optimizer (SGO): a novel meta-heuristic algorithm. Sci. Rep. **13**(1), 5373 (2023)
63. Mehrabian, A.R., Lucas, C.: A novel numerical optimization algorithm inspired from weed colonization. Eco. Inform. **1**(4), 355–366 (2006)
64. Ma, L., Hu, K., Zhu, Y., Chen, H., & He, M.: A novel plant root foraging algorithm for image segmentation problems. Math. Probl. Eng. **2014**(1), 471209 (2014)
65. Yang, X. S.: Flower pollination algorithm for global optimization. In International conference on unconventional computing and natural computation (pp. 240–249). Springer, Berlin (2012).
66. Chandra, S.S., Hareendran S, A.: Phototropic algorithm for global optimisation problems. Appl. Intell. **51**(8), 5965–5977 (2021)
67. Abdelhamid, A.A., Towfek, S.K., Khodadadi, N., Alhussan, A.A., Khafaga, D.S., Eid, M.M., Ibrahim, A.: Waterwheel plant algorithm: a novel metaheuristic optimization method. Processes **11**(5), 1502 (2023)
68. Hatamlou, A.: Black hole: a new heuristic optimization approach for data clustering. Inf. Sci. **222**, 175–184 (2013)
69. Zheng, Y.J.: Water wave optimization: a new nature-inspired metaheuristic. Comput. Oper. Res. **55**, 1–11 (2015)
70. Shareef, H., Ibrahim, A.A., Mutlag, A.H.: Lightning search algorithm. Appl. Soft Comput. **36**, 315–333 (2015)
71. Abedinpourshotorban, H., Shamsuddin, S.M., Beheshti, Z., Jawawi, D.N.: Electromagnetic field optimization: a physics-inspired metaheuristic optimization algorithm. Swarm Evol. Comput. **26**, 8–22 (2016)
72. Mirjalili, S.: SCA: a sine cosine algorithm for solving optimization problems. Knowl.-Based Syst. **96**, 120–133 (2016)
73. Kaveh, A., Dadras, A.: A novel meta-heuristic optimization algorithm: thermal exchange optimization. Adv. Eng. Softw. **110**, 69–84 (2017)
74. Daliri, A., Asghari, A., Azgomi, H., Alimoradi, M.: The water optimization algorithm: a novel metaheuristic for solving optimization problems. Appl. Intell. **52**(15), 17990–18029 (2022)
75. Faramarzi, A., Heidarinejad, M., Stephens, B., Mirjalili, S.: Equilibrium optimizer: a novel optimization algorithm. Knowl.-Based Syst. **191**, 105190 (2020)
76. Abdel-Basset, M., Mohamed, R., Sallam, K.M., Chakrabortty, R.K.: Light spectrum optimizer: a novel physics-inspired metaheuristic optimization algorithm. Mathematics **10**(19), 3466 (2022)

77. Kundu, R., Chattopadhyay, S., Nag, S., Navarro, M.A., Oliva, D.: Prism refraction search: a novel physics-based metaheuristic algorithm. J. Supercomput. **80**(8), 10746–10795 (2024)
78. Deng, L., Liu, S.: Snow ablation optimizer: a novel metaheuristic technique for numerical optimization and engineering design. Expert Syst. Appl. **225**, 120069 (2023)
79. Zhang, H., San, H., Sun, H., Ding, L., Wu, X.: A novel optimization method: wave search algorithm. J. Supercomput., pp 1–36 (2024).
80. Iben, I., Jr.: Stellar evolution within and off the main sequence. Ann. Rev. Astron. Astrophys. **5**(1), 571–626 (1967)
81. Bromm, V., Yoshida, N., Hernquist, L., McKee, C.F.: The formation of the first stars and galaxies. Nature **459**(7243), 49–54 (2009)
82. Luhman, K.L.: The formation and early evolution of low-mass stars and brown dwarfs. Ann. Rev. Astron. Astrophys. **50**, 65–106 (2012)
83. Hekker, S., Christensen-Dalsgaard, J.: Giant star seismology. Astron. Astrophys. Rev. **25**, 1–122 (2017)
84. Kwitter, K.B., Henry, R.B.C.: Planetary nebulae: sources of enlightenment. Publ. Astron. Soc. Pac. **134**(1032), 022001 (2022)
85. Saumon, D., Blouin, S., Tremblay, P.E.: Current challenges in the physics of white dwarf stars. Phys. Rep. **988**, 1–63 (2022)
86. Amaro, D., Faraji, S.: Vacuum polarization effects in the background of a deformed compact object and implications for photon velocity. Phys. Rev. D **111**(4), 045003 (2025)
87. Surjanovic, S., Bingham, D.: Virtual Library of simulation experiments: test functions and datasets. Retrieved October 23, 2017, from http://www.sfu.ca/∼ssurjano (2013)
88. Kumar, A., Wu, G., Ali, M.Z., Mallipeddi, R., Suganthan, P.N., Das, S.: A test-suite of non-convex constrained optimization problems from the real-world and some baseline results. Swarm Evol. Comput. **56**, 100693 (2020)
89. Kumar, A., Das, S., Zelinka, I.: A self-adaptive spherical search algorithm for real-world constrained optimization problems. In: Proceedings of the 2020 genetic and evolutionary computation conference companion, (pp. 13–14) (2020).
90. Gurrola-Ramos, J., Hernàndez-Aguirre, A., Dalmau-Cedeño, O.: COLSHADE for real-world single-objective constrained optimization problems. In: 2020 IEEE congress on evolutionary computation (CEC) (pp. 1–8). IEEE, New York (2020).
91. Kumar, A., Das, S., Zelinka, I.: A modified covariance matrix adaptation evolution strategy for real-world constrained optimization problems. In: Proceedings of the 2020 Genetic and Evolutionary Computation Conference Companion (pp. 11–12) (2020).
92. Harifi, S., Davachi, F., Mohammadi, N., FaridMohammadzadegan, S.: Two competitive hybridization approaches based on combining of Giza Pyramids Construction with Particle Swarm Optimization for solving global optimization problems. Intel. Artif. **28**(75), 114–139 (2025)
93. Ebneyousef, S., Shirmarz, A.: A taxonomy of load balancing algorithms and approaches in fog computing: a survey. Clust. Comput. **26**(5), 3187–3208 (2023)
94. Pourghebleh, B., Hayyolalam, V.: A comprehensive and systematic review of the load balancing mechanisms in the Internet of Things. Clust. Comput. **23**(2), 641–661 (2020)
95. Ghomi, E.J., Rahmani, A.M., Qader, N.N.: Load-balancing algorithms in cloud computing: a survey. J. Netw. Comput. Appl. **88**, 50–71 (2017)
96. Pradhan, A., Bisoy, S.K.: A novel load balancing technique for cloud computing platform based on PSO. J. King Saud Univ.-Comput. Inform. Sci. **34**(7), 3988–3995 (2022)
97. Wang, C., Zhang, G., Xu, H., & Chen, H. (2016, November). An ACO-based link load-balancing algorithm in SDN. In 2016 7th International Conference on Cloud Computing and Big Data (CCBD) (pp. 214–218). IEEE, New York.
98. Fu, X., Sun, Y., Wang, H., Li, H.: Task scheduling of cloud computing based on hybrid particle swarm algorithm and genetic algorithm. Clust. Comput. **26**(5), 2479–2488 (2023)
99. Ullah, A., Nawi, N.M., Uddin, J., Baseer, S., Rashed, A.H.: Artificial bee colony algorithm used for load balancing in cloud computing. IAES Int. J. Artif. Intell. **8**(2), 156 (2019)
100. Ullah, A., Nawi, N.M., Khan, M.H.: BAT algorithm used for load balancing purpose in cloud computing: an overview. Int. J. High Perform. Comput. Networking **16**(1), 43–54 (2020)
101. Chen, X., Cheng, L., Liu, C., Liu, Q., Liu, J., Mao, Y., Murphy, J.: A WOA-based optimization approach for task scheduling in cloud computing systems. IEEE Syst. J. **14**(3), 3117–3128 (2020)
102. Mondal, B., Choudhury, A.: Simulated annealing (SA) based load balancing strategy for cloud computing. Int. J. Comput. Sci. Inform. Technol. **6**(4), 3307–3312 (2015)
103. Ebadifard, F., Babamir, S.M.: Autonomic task scheduling algorithm for dynamic workloads through a load balancing technique for the cloud-computing environment. Clust. Comput. **24**, 1075–1101 (2021)

Sasan Harifi received the B.Sc. degree in computer engineering from the Islamic Azad University of Karaj, Karaj, Iran, in 2011, the M.Sc. degree in software engineering from the Faculty of Electrical, Computer and IT Engineering, Islamic Azad University, Qazvin Branch, Iran, in 2015, and the Ph.D. degree in software systems from the Islamic Azad University of Karaj, Karaj, Iran, in 2019. He is currently an Assistant Professor at the Department of Computer Engineering, Islamic Azad University, Karaj, Iran. His current research interests include optimization algorithms, metaheuristic algorithms, swarm intelligence, ancient inspired computing, nature inspired computing, machine learning, and deep learning. He is also the first to introduce a novel source of inspiration called "Ancient-inspired" to develop metaheuristic algorithms.

Reza Eghbali received the M.Sc. degree in computer science (artificial intelligence) from Kharazmi University, Tehran, Iran, in 2017. He is currently working toward the Ph.D. degree in computer science (artificial intelligence) with the Islamic Azad University of Karaj, Iran. His research interests include machine learning (deep, reinforcement, and federated learning), metaheuristic algorithms, multi-agent systems, and edge computing.

Seyed Mohsen Mirhosseini received the Ph.D. in computer engineering from Shahid Beheshti University. He is currently an Assistant Professor at the Faculty of Artificial Intelligence, Islamic Azad University, Karaj, Iran. His research interests include software testing, metaheuristic algorithms, and machine learning.