Neural Computing and Applications
https://doi.org/10.1007/s00521-021-06392-x

ORIGINAL ARTICLE

A novel meta-heuristic algorithm for solving numerical optimization problems: Ali Baba and the forty thieves

Malik Braik[1] · Mohammad Hashem Ryalat[1] · Hussein Al-Zoubi[2]

Received: 29 November 2020 / Accepted: 26 July 2021

Abstract
This paper presents a novel meta-heuristic algorithm called Ali Baba and the forty thieves (AFT) for solving global optimization problems. Recall the famous tale of Ali Baba and the forty thieves, where Ali Baba once saw a gang of forty thieves enter a strange cave filled with all kinds of treasures. The strategies pursued by the forty thieves in the search for Ali Baba inspired us to design ideas and underlie the basic concepts to put forward the mathematical models and implement the exploration and exploitation processes of the proposed algorithm. The performance of the AFT algorithm was assessed on a set of basic benchmark test functions and two more challenging benchmarks called IEEE CEC-2017 and IEEE CEC-C06 2019 benchmark test functions. These benchmarks cover simple and complex test functions with various dimensions and levels of complexity. An extensive comparative study was performed between the AFT algorithm and other well-studied algorithms, and the significance of the results was proved by statistical test methods. To study the potential performance of AFT, its further development is discussed and carried out from five aspects. Finally, the applicability of the AFT algorithm was subsequently demonstrated in solving five engineering design problems. The results in both benchmark functions and engineering problems show that the AFT algorithm has stronger performance than other competitors' algorithms.

Keywords Optimization · Meta-heuristics · Nature-inspired algorithms · Ali Baba and the forty thieves algorithm · Optimization techniques

1 Introduction

Solving optimization problems is the norm in almost all disciplines of engineering [1, 2] and science [3, 4], and the need for more robust solutions is ever increasing. This means, we need plausible algorithms that can fit the intricate nature of such up-to-date scientific and engineering challenges. When surveying the literature for the existing optimization methods, one may find a wide variety of these methods [5]. These range from traditional optimization techniques that use both linear and nonlinear programming [6], to the newer nature-inspired meta-heuristics [7, 8], each with their own strengths and weaknesses. Despite being successful in solving well-known optimization problems [2, 9], traditional algorithms on one side suffer from inherent dependency on gradient information and the desperate need for a promising initial starting vector within the search space [2, 9]. The existing nature-inspired meta-heuristic optimizers, on the other side, are highly problem dependent in that they might be very successful in solving certain problems and may not be able to provide satisfactory solutions for other problems [10]. This is partly ascribed to the common behavior of these meta-heuristics of being trapped in local or suboptimal solutions [11].

What makes contemporary optimization problems tough in nature can be summarized in a few points. It is very likely that such problems are nonlinear in essence, troublesome in nature, contain numerous decision variables,

✉ Malik Braik
mbraik@bau.edu.jo

Mohammad Hashem Ryalat
ryalat@bau.edu.jo

Hussein Al-Zoubi
halzoubi@yu.edu.jo

[1] Al-Balqa Applied University, Al-Salt, Jordan
[2] Yarmouk University, Irbid, Jordan

Published online: 09 August 2021

their objective functions are, in some cases, complex and handcuffed with several constraints, in addition to having multifarious peaks [12]. For these problems, it is imperative to go ahead from an encouraging starting point with the hope of finally hitting the global optimum solution. After many years of research, the research community has found that traditional methods might not represent themselves as the best catch for solving these contemporaneous optimization problems [13]. More specifically, the found solution must be accurate enough to be accepted, and the time needed to solve the problem should fall within reasonable ranges [14]. To this end, the researchers have turned their attention toward nature-inspired meta-heuristics that have shown extremely heartening capabilities in dealing with very knotted shapes of challenging optimization problems [15, 16]. Meta-heuristic techniques are global optimization methods designed based on simulations and methods inspired by nature that are openly applied to solve global optimization problems [17, 18].

In contrast to traditional algorithms, meta-heuristic methods have become startlingly very trendy. This reputation is due to the fact that these methods are very flexible, do not require gradient information and have proven their success in escaping from local minimums when solving real-world scientific or engineering problems that have several local solutions [11, 19]. It is important to note that the first and second merits stand out from the verity that meta-heuristics tackle optimization problems by assuming them as black boxes in that they only require knowledge of the input and output sets of the variables. Thereby, there is no necessity to calculate a derivative of the search space. Also, meta-heuristics belong to the family of stochastic optimization methods, in which they make use of the stochastic operators. This feature has been broadly affirmed, in which meta-heuristics have proven successful in keeping away from local minima when addressing real problems that often have a large number of local minimums [11, 18]. This explains their eligibility to handle challenging optimization problems in diversified domains [20, 21]. More closely, meta-heuristics have been harnessed to tackle hard real-life problems in a variety of scientific and engineering disciplines. Examples of such domains encompass, but are not limited to, image processing [22, 23], signal processing [24], the realm of process control [25], text clustering [26], classification problems [27] as well as several other domains [28, 29].

1.1 Motivations of the work

According to the "no-free-lunch" (NFL) theory [30], it is difficult to employ a single meta-heuristic algorithm in striving to solve all possible optimization problems [31]. As it really is, one meta-heuristic algorithm might do a good job in optimizing certain problems in particular fields, but it falls short to find the global optima in other fields [11]. This has been a motive for the researchers in this field, as well as ourselves, to look for new and innovative nature-inspired methods to solve and show superior scores on the current and new hard real-life problems [32]. The door is still open, and here we present a novel meta-heuristic algorithm based on human behavior with the very famous tale of Ali Baba and the forty thieves, as our inspiration targeting numerical optimization problems.

1.2 Contributions of the work

The core of this paper is to establish a novel nature-inspired algorithm, referred to as Ali Baba and the forty thieves (AFT), to solve global optimization problems. As its name glimpses, AFT falls into the category of human-based algorithms, as it is inspired by human interactions and human demeanor in a human-related story. The thieves' behavior, in the tale of Ali Baba and the forty thieves, in finding Ali Baba and the intelligent methods that Ali Baba's maid used to save him from the thieves, inspired us to simulate this behavior with an optimization algorithm. In this anecdote, there is a behavior that has many similarities with optimization processes. From the point of view of optimization, the thieves are the search agents, the environment (i.e., town of Ali Baba) is the search space, each position of the town corresponds to a realizable solution, the home of Ali Baba is the objective function and Ali Baba is the global solution. Based on these similarities, the AFT algorithm was developed to mimic the behavior of the thieves and the maid to locate the global solution to the considered optimization problems. The performance of AFT was evaluated on sixty-two benchmark test functions, and it was applied to optimize the designs of five engineering problems.

Section 2 presents the literature and related works. Section 3 shows the tale of Ali Baba and the forty thieves and the key concepts of this tale. Section 4 presents the mathematical models and analysis of the AFT method. Some of the possible expansions of AFT from several aspects are given in Sect. 5. Section 6 then presents a conceptual comparison of AFT with other existing optimizers. The experimental, qualitative and statistical analysis results are introduced in Sect. 7. Section 8 presents the applicability and reliability of AFT in solving five engineering problems. The conclusion comments and some further research paths are shown in Sect. 9.

2 Related works

This section looks at the most recent developments in the field of optimization. There are many sections in this field such as multi-objective, single-objective, constrained and others [33]. Since the meta-heuristic algorithm proposed in this work is turned to solve single optimization problems, the chief hub of this section concerns the relevant works in single optimization areas.

2.1 Single-objective optimization problems

In single-objective optimization problems, there is only one objective to be maximized or minimized. This kind of optimization might be subject to a set of constraints, which fall into two categories: equality and inequality [11]. Without loss of generality, single-objective optimization problems can be expressed as a minimization or a maximization problem. The search space is created using a set of variables, objectives, range of variables and constraints. For optimization problems, the search space can be easily plotted in the Cartesian coordinate system and its shapes can then be observed. Having a large number of decision variables is the first challenge when addressing optimization problems. The limitation of the search space is the range of variables, which is diversified. These variables can be discrete or continuous. This means that they either create a discrete or a continuous search space. In the first case, there is a finite set of points between two points in the search space, while in the second case, there is an infinite number of points between every two points [11].

Usually, an optimization method might begin with an initial range and extend it while optimization. The constraints restrict the search space even more, and typically lead to breaks in the search space because the solutions in those areas are not appropriate when solving an optimization problem. A set of constraints can even divide the search space into various detached areas. The solutions that penetrate the constrained regions are named infeasible solutions, while the solutions in the constrained regions are named feasible solutions. There are two terms for the portions of the search space that are within and out of the constrained regions: feasible and infeasible. A restricted search space has the potency to render the optimization method ineffective in spite of its sensible performance in an unrestricted search space [34]. Thus, optimization methods must be well prepared with adequate operators to deal effectively with the constraints [34]. Another challenge that arises when tackling optimization problems is the existence of local solutions.

In a single-objective search space, there is usually the global optimal solution that returns the best objective value. However, there are normally several other solutions that yield values close to the objective value of the global optimal [33]. This type of solutions is named local solutions as it is locally the best solution if we take into account the search space in its vicinity. On the other hand, it is not the best solution globally when taking into account the whole search space. The existence of local solutions in optimization problems bring many optimization algorithms to fall into local solutions [8]. A real search space generally contains a large number of local solutions. Thus, an optimization method must be able to efficiently averting them to find the global optimum. An optimization algorithm that is able to eschew local solutions is not necessarily capable of converging to the global optimum. The approximate position of the global optimal is found when an algorithm averts local solutions. The convergence speed is also a difficulty when solving optimization problems [8]. Needless to say, rapid convergence leads to local optima stagnation. In contrast, abrupt variations in the solutions result in avoiding local optima, but slow down the convergence rate toward the global optimal. These two trade-offs are the key challenges that an optimization algorithm handles while addressing optimization problems. There are other varieties of difficulties when addressing a single-objective search space such as: isolation of the optimum, dynamic objective function and many more [11]. Each of these challenges demands special attention. These conceptions are outside the scope of this paper, so solicitous readers are referred to the studies conducted by Boussaid [33].

2.2 Single-objective optimization algorithms

In the literature, optimization algorithms can be split into two broad categories:

- *Deterministic algorithms* these algorithms always locate the same solution for a particular problem if they commence with the same starting point. The main merit of these methods is the reliability as they decidedly find a solution in each run. However, local optima stagnancy is a flaw as these algorithms do not typically contain random behaviors when solving optimization problems.
- *Stochastic algorithms* these algorithms benefit from stochastic operators. This leads to find a different solution at each run even if the starting point, in the runs, remains unaltered and thus makes stochastic methods less reliable as compared to the deterministic methods. However, the stochastic behavior has the vigor to avoid the local optimums. The reliability of stochastic algorithms can be boosted by adjusting and rising the number of runs. Stochastic methods fall into two classes:

2.2.1 Individualist algorithms

The stochastic method starts and carries out optimization with a single solution. This solution is randomly changed and enhanced for a predefined number of steps or realization of a final criterion. The most well-respected algorithms in this class are Tabu search [35], hill climbing [36] and iterated local search [37]. The most chief feature of the algorithms in this set is the low computational effort and the need for few function evaluations.

2.2.2 Collective algorithms

Collective techniques generate and develop multiple random solutions during optimization. Usually, the collection of solutions collaborates to better identify the global optimum in the search domain. Multiple solutions reduce the chance to slack in local optima [38]. This is a key merit of these algorithms. However, each of the solutions requires a single function evaluation, where building an efficient cooperation between the solutions is a challenge. Despite these two flaws, collective stochastic optimization methods are widely used in optimization problems [11]. This is due to the well coveted features of these algorithms.

Irrespective of the distinctions between collective algorithms, they all pursue the same course of action for finding the global optimum. Optimization first begins with a pool of random solutions, which need to be combined and changed at random, quickly, and suddenly. This elicits that the solutions move globally. This stage is called *exploration* of the search space because the solutions are attracted toward various areas of the search space by abrupt changes [38]. After sufficient exploration, the solutions begin to sparingly change and move locally around the most promising solutions of the search space in order to raise their quality. This phase is called *exploitation*, and its key aim is to enhance the precision of the best solutions got in the exploration phase [39]. Although avoidance of local optima may occur in the exploitation phase, the coverage of search area is not as broad as the exploration occur. In this case, the solutions evade local solutions in the proximity of the global optimal. So we can deduce that the exploration and exploitation phases pursue inconsistent goals [40]. So, most of the methods seamlessly demand the search agents to transit from exploration to exploitation using adaptive strategies. A convincing recommendation for good performance is to achieve an adequate balance between them [38]. Due to the random behavior of the meta-heuristics, they can be deemed as stochastic collective algorithms [21, 41]. Continuing from this last point, any meta-heuristic algorithm can fall into:

- *Physics-Based (PB) algorithms* these methods utilize the physical foundations present on Earth, in particular, or in our universe, at the broadest level. The general technique of PB methods is different from other meta-heuristics' mechanisms because the search agents of these methods contact around the search space according to physics rules firmed up in physical processes. Some of the most prominent examples of PB algorithms include simulated annealing (SA) algorithm [42, 43], gravitational search algorithm (GSA) [44], multi-verse optimizer (MVO) [45], Henry gas solubility optimization (HGSO) [46] and equilibrium optimizer (EO) [47].
- *Evolutionary Algorithms (EAs)* EAs follow the Darwinian theory of natural selection and biological Darwinism that represents the survival of the fittest, where the well-known evolution mechanisms in biology are simulated. These methods often do well at finding near-optimal solutions in view of the fact that they do not lend any credence about the underlying fitness landscape. The list of EAs includes, but not limited to, evolutionary strategy (ES) [48], genetic algorithm (GA) [49, 50], genetic programming (GP) [51] and differential evolution (DE) algorithm [52].
- *Swarm Intelligence (SI) algorithms* these algorithms use the intelligence of the social collective behavior of various societies of creatures such as birds, bees, ants and the alike. This class includes a large variety of algorithms such as particle swarm optimization (PSO) [53], ant colony optimization (ACO) [54], artificial bee colony (ABC) algorithm [55], grey wolf optimizer (GWO) [56], dragonfly algorithm (DA) [57], salp swarm algorithm [11, 58], coral reefs optimization (CRO) [59] and many others [7, 8].
- *Human-based algorithm* the algorithms of this class originate from human interactions in societies. The inspiration for researchers in the realm of human-based algorithms comes from experiences and stories related to human demeanor and human actions [60]. Previous works in this area include harmony search (HS) [61], seeker optimization algorithm [62], Human Group Formation (HGF) [63], Social-Based Algorithm (SBA) [64], Interior Search Algorithm (ISA) [65] and the football game inspired algorithm (FGIA) [66].

3 Inspiration

The proposed AFT algorithm is based on the well-known tale of Ali Baba and the forty thieves. We have found in this anecdote several intrinsic traits that inspired us to develop the AFT algorithm. Rather than literally retelling the story in this section, we prefer to link some of the events that took place in the tale to the attributes that constitute the AFT algorithm. The tale itself can be found in several books as well as web pages. We refer the interested readers to [67], as an example.

The nature of the tale is a search-based, where a gang of forty thieves go after Ali Baba. The ultimate goal of the gang is to catch Ali Baba for revenge and to get their treasure back. The search carried out by the gang for Ali Baba is iterative in nature, going in several rounds, each time reinforcing the solution found by previous iterations. The search is based upon the collective behavior of the thieves, represented as a population in the proposed algorithm. The counter measures took by the main character of the tale, named Marjaneh, prevented the gang at each iteration from fulfilling their search mission. The big town, where Ali Baba lives, represents the search space. The tale shows the success of the forty thieves in tracking down Ali Baba and spotting the location of his house. The looters' successful actions in achieving their strategic target and finding Ali Baba were based on smart tricks and tactics. However, the acumen of the savvy maid of Ali Baba, Marjaneh, saved the life of Ali Baba in each of these tactics, abridged as follows [68].

In the first trial, the gang's assistant captain disguised himself as a foreigner and entered the town in the hope to hear any talk or find any clue that could lead him to Ali Baba. He managed to get to Ali Baba's house and marked his door with an 'X' sign in his pursuit to later fulfill the ultimate mission of the gang. Marjaneh observed the mark and, in response, she placed similar marks on the doors of all houses in the neighborhood, rendering the plan useless. The tactics followed by the robber are harnessed in the proposed algorithm to maximize the exploration efficiency.

The second trial took place when another assistant of the captain took the mission. He built upon the procedures previously followed by his comrade. In our algorithm, this is used by making the search in every iteration builds upon the best solution found so far from previous iterations. This time, the robber marked Ali Baba's house with a sign that is not easy to be observed by chance. In AFT, this again reflects on the utilization and enhancement of previously found solutions [69]. This leads to strong exploration and exploitation features in the proposed algorithm.

The third incident occurred when the captain decided to change the plan, and take upon himself the task of capturing Ali Baba, yet to build upon the achievements attained so far by his two assistants. The captain and his followers succeeded in arriving at Ali Baba's house, and they took every possible measure not to be discovered while attacking Ali Baba, but they failed.

The final trial in targeting Ali Baba took place by the captain alone with a totally new plan, who disguised as a merchant selling silks, and introduced himself to the son of Ali Baba. The perseverance and persistence of the gang's captain are good traits for a successful search technique.

The approaches adopted in the tale, such as the attempts of the thieves and Marjana's intelligence to disrupt these attempts, are reflected in the exploration and exploitation mechanisms built into the proposed algorithm. This has led to the mathematical models developed to design AFT and perform optimization. The proposed algorithm is described in detail below.

4 Ali Baba and the forty thieves algorithm

The overall goal of this work is to present a new optimization method that imitates the tale of Ali Baba and the forty thieves as a coordinated model of social behavior of humans' actions. The following principles derived from this tale achieve the basic assumptions of this algorithm:

- The forty thieves collaborate in a group and get guidance from someone or from one of the thieves to find Ali Baba's house. This information may or may not be correct.
- The forty thieves will travel a distance starting from an initial distance until they can locate Ali Baba's house.
- Marjaneh can deceive the thieves many times with astute ways to somehow protect Ali Baba out of arrival of them by a proportion.

The behaviors of the thieves and Marjaneh can be drawn up in such a manner that they can be linked to an objective function to be optimized. This makes it feasible to evolve a new meta-heuristic algorithm as detailed below.

4.1 Random initialization

The AFT algorithm is initiated by randomly initializing the position of a number of n individuals in a d-dimensional search space as shown below:

$$x = \begin{bmatrix} x_1^1 & x_2^1 & x_3^1 & \dots & x_d^1 \\ x_1^2 & x_2^2 & x_3^2 & \dots & x_d^2 \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ x_1^n & x_2^n & x_3^n & \dots & x_d^n \end{bmatrix} \quad (1)$$

where x is the position of all thieves, d is the number of variables of a given problem and x_j^i represents the jth dimension of the ith thief.

The initial position of population (i.e., thieves) can be generated as shown in Eq. 2.

$$x^i = l_j + r \times (u_j - l_j) \tag{2}$$

where x^i is the position of the ith thief that denotes a candidate solution to a problem, l_j and u_j refer to the lower and upper bounds in the jth dimension, respectively, and r is a uniformly distributed random number in the range from 0 to 1.

The wit level of Marjaneh with respect to all thieves can be initialized as shown below:

$$m = \begin{bmatrix} m_1^1 & m_2^1 & \dots & m_d^1 \\ m_1^2 & m_2^2 & \dots & m_d^2 \\ \vdots & \vdots & \vdots & \vdots \\ m_1^n & m_2^n & \dots & m_d^n \end{bmatrix} \tag{3}$$

where m_j^i denotes the astute level of Marjaneh in relation to the ith thief at the jth dimension.

4.2 Fitness evaluation

The values of the decision variables are inserted into a user-defined fitness function that is evaluated for each thief's position. The corresponding fitness values are stored in an array as given in the following form:

$$f = \begin{bmatrix} f_1([x_1^1, & x_2^1, & \dots, & x_d^1)] \\ f_2([x_1^2, & x_2^2, & \dots, & x_d^2)] \\ \vdots & \vdots & \vdots & \vdots \\ f_n([x_1^n, & x_2^n, & \dots, & x_d^n)] \end{bmatrix} \tag{4}$$

where x_d^n is the dth dimension of the position of the nth thief.

In the simulation of the AFT algorithm, the solution quality is evaluated for each thief's new location based upon a defined fitness function. After that, the location is updated if it is better than the solution quality of the current one. Each thief stays in his current location if his solution quality is more efficient than the new one.

4.3 Proposed mathematical model

As discussed above, three fundamental cases may occur while thieves search for Ali Baba. In each case, it is assumed that the thieves search efficiently throughout the surrounding environment, while there also a proportion that occurs due to Marjaneh's intelligence that forces the thieves to search in random locations. The above searching behavior can be mathematically modeled as follows:

Case 1 The thieves may track down Ali Baba with the help of information obtained from someone. In this case, the new locations of the thieves can be obtained as follows:

$$\begin{aligned} x_{t+1}^i =& gbest_t + \Big[Td_t \left(\text{best}_t^i - y_t^i\right) r_1 \\ & + Td_t \left(y_t^i - m_t^{a(i)}\right) r_2 \Big] \operatorname{sgn}(\text{rand} - 0.5); \\ & r_3 \geq 0.5, \quad r_4 > Pp_t \end{aligned} \tag{5}$$

where x_{t+1}^i represents the position of thief i at iteration $(t+1)$, y_t^i is the position of Ali Baba in relation to thief i at iteration t, best_t^i represents the best position that has achieved so far by thief i at iteration t, $gbest_t$ represents the global best position obtained so far by any thief up to the t_{th} iteration, $m_t^{a(i)}$ represents Marjaneh's intelligence level used to camouflage thief i at iteration t, Td_t is the tracking distance of the thieves at iteration t, Pp_t denotes the perception potential of the thieves to Ali Baba at iteration t, rand, r_1, r_2 and r_4 are random numbers generated with a uniform distribution between 0 and 1 , $r_3 \geq 0.5$ gives either 1 or 0 to indicate that the information obtained to the thieves is true or false, respectively, and $\operatorname{sgn}(\text{rand} - 0.5)$ gives either 1 or $-$ 1 to change the direction of the search process.

The parameter a in $m_t^{a(i)}$ can be defined as follows:

$$a = \lceil (n-1) \cdot \text{rand}(n, 1) \rceil \tag{6}$$

where rand(n, 1) represents a vector of random numbers generated with a uniform distribution in the range of [0, 1].

Marjaneh updates her astute plans if the quality of the new solution that the thieves come up with is better than their previous position. In this case, Eq. 7 can be used to update her's plans.

$$m_t^{a(i)} = \begin{cases} x_t^i & \text{if } f\left(x_t^i\right) \geq f\left(m_t^{a(i)}\right) \\ m_t^{a(i)} & \text{if } f\left(x_t^i\right) < f\left(m_t^{a(i)}\right) \end{cases} \tag{7}$$

where $f(\cdot)$ stands for the score of the fitness function.

The tracking distance parameter Td_t is defined as given in Eq. 8.

$$\text{Td}_t = \alpha_0 e^{-\alpha_1 (t/T)^{\alpha}_1} \tag{8}$$

where t and T denote the current and maximum number of iterations, respectively, α_0 ($\alpha_0 = 1$) represents the initial estimate of the tracking distance at the first iteration and α_1

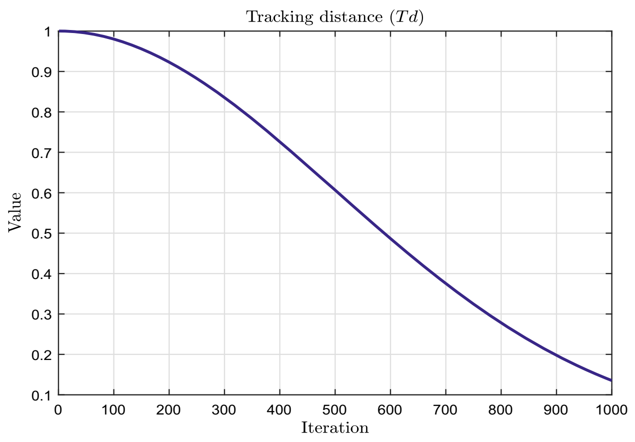

Fig. 1 Proposed exponential iterative function for Td_t

is a constant value used to manage exploration and exploitation capabilities.

Equation 8 shows that Td_t is iteratively updated during the course of iterations of the AFT algorithm. Figure 1 shows the values of Td_t over 1000 iterations.

The tracking distance, as shown in Fig. 1, greatly affects the search ability, which has a large impact on both the exploration and exploitation power of the AFT algorithm. As presented in Fig. 1, the parameter Td_t starts from a value of 1.0 and goes down to the lowest value where it assumes that the thieves have arrived at Ali Baba's house. Large values of Td_t result in global search that can be diverted toward further exploration, and this may avoid local optimal solutions. On the other side, small values of Td_t lead to local search, where this increases the exploitation ability in AFT so that the thieves have a good possibility to find Ali Baba.

Similarly, the perception potential parameter Pp_t was defined as given in Eq. 9.

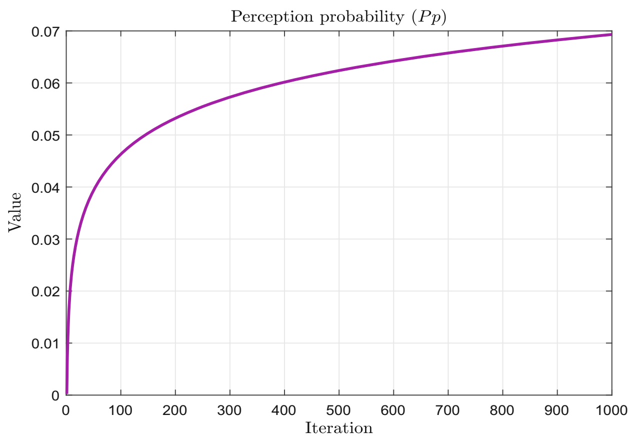

Fig. 2 Proposed exponential iterative function for Pp_t

$$\mathrm{Pp}_t = \beta_0 log(\beta_1 (t/T)^{\beta_0}) \qquad (9)$$

where β_0 ($\beta_0 = 0.1$) represents a final rough estimation of the probability that the thieves will realize their target at the end of the iterative process of AFT and β_1 is a constant value used to manage exploration and exploitation capabilities.

Figure 2 shows the values of Pp_t over 1000 iterations.

As shown in Fig. 2, by gradually increasing the value of Pp_t, AFT tends to move from global search to local search in the most promising areas where a potential solution could be found in these areas. In other words, large values of Pp_t lead to local search that intensifies the search in the most appropriate areas of the search space. On the other side, small values reduce the possibility of searching in the vicinity of current good solutions. Thus, an increase in this value stimulates AFT to explore the search space on a global scale and to diversify the search in all areas of the search space.

For all of the problems solved in this work, α_1 and β_1 are both equal 2.0. These parameters are found by experimental testing for a large subset of test functions.

Case 2 The thieves may grasp that they have been deceived, so they will randomly explore the search space for Ali Baba. In this case, the new locations of the thieves can be obtained as follows:

$$x^i_{t+1} = \mathrm{Td}_t\left[\left(u_j - l_j\right)\mathrm{rand} + l_j\right]; \quad r_3 \geq 0.5, \ r_4 \leq \mathrm{Pp}_t \qquad (10)$$

The parameter Td_t is incorporated in Eq. 10 because the thieves have a good level of knowledge to discern of the most propitious areas of the search space where Ali Baba's house could be.

Case 3 In order to ameliorate the exploration and exploitation features of the AFT algorithm, this study also considers the search in other positions than those that could be obtained using Eq. 5. In this case, the new locations of the thieves can be obtained as follows:

$$\begin{aligned} x^i_{t+1} = & gbest_t - \Big[Td_t\left(\mathrm{best}^i_t - y^i_t\right)r_1 \\ & + Td_t\left(y^i_t - m^{a(i)}_t\right)r_2\Big] \ \mathrm{sgn}(\mathrm{rand} - 0.5); \\ & r_3 < 0.5 \end{aligned} \qquad (11)$$

The pseudo-code of the AFT algorithm can be briefly described by the iterative steps given in Algorithm 1.

Algorithm 1 A pseudo-code presents a summary of the key steps of the AFT algorithm.

```
1: $u_j$ and $l_j$ are the upper and lower bounds of the search space in the $j$th dimension
2: Randomly initialize the position, $x$, of all $n$ thieves in the search space
3: Initialize the best position ($best_t^i$) and global best position ($gbest_t$) for the thieves
4: Initialize the intelligence degree of Marjaneh with respect to all thieves
5: Evaluate the position of all thieves using a fitness function ($f(x)$)
6: Set $t \leftarrow 1$
7: while ($t < T$) do
8:     $Td_t = 1.0 \times e^{-2.0(t/T)^{2.0}}$
9:     $Pp_t = 0.1 \times log(2.0(t/T)^{0.1})$
10:    for $i = 1, 2, \ldots, n$ do
11:        if ($rand \geq 0.5$) then
12:            if ($rand \geq Pp_t$) then
13:                $x_{t+1}^i = gbest_t + \left[Td_t\left(best_t^i - y_t^i\right)r_1 + Td_t\left(y_t^i - m_t^{a(i)}\right)r_2\right]\text{sgn}(rand - 0.5)$
14:            else
15:                $x_{t+1}^i = Td\left[(u_j - l_j)\,rand + l_j\right]$
16:            end if
17:        else
18:            $x_{t+1}^i = gbest_t - \left[Td_t\left(best_t^i - y_t^i\right)r_1 + Td_t\left(y_t^i - m_t^{a(i)}\right)r_2\right]\text{sgn}(rand - 0.5)$
19:        end if
20:    end for
21:    for $i = 1, 2, \ldots, n$ do
22:        Check the feasibility of new positions
23:        Evaluate and update the new position of the thieves
24:        Update the solutions $best_t^i$ and $gbest_t$
25:        Update $m_t^{a(i)}$ using Equation 7
26:    end for
27:    $t = t + 1$
28: end while
```

Algorithm 1 reveals that AFT initiates optimization in solving an optimization problem by randomly generating a set of positions (i.e., potential solutions), considering the upper and lower bounds of the problem variables. After that, the best position, the global best position of the thieves and Marjaneh's wit plans are initialized. The quality of each created solution is assessed using a predefined fitness function, whereby the suitability of each solution is recalculated within each iteration in order to identify the thief with the optimal solution. For each dimension, the new position of the thieves is computed iteratively within each iteration using Eqs. 5, 10 and 11. The feasibility of each new position is examined to see if it moves out of the search area. In such a context, it will be brought back to the boundary on the basis of the simulated steps of AFT. Then, the new position, the best position, the global best position of the thieves and the wit plans of Marjaneh are assessed and updated accordingly. All the steps of AFT shown in Algorithm 1, except the initialization steps, are iteratively performed until the termination evaluation condition is reached. At the end, the best position of the thieves is scored as a solution of the optimization problem.

4.4 Exploration ability of AFT

There are many parameters in AFT that lead to exploration, explained as follows:

- Td_t: this parameter controls the exploration quantity of AFT. It identifies the extent to which the new locations of the thieves would be to the house of Ali Baba. The selection of appropriate values for α_0 and α_1 for Td_t would reduce the recession probability in local optima and augment the probability of approaching the global optimum. Based on the experimental tests, $\alpha_0 = 1$ and $\alpha_1 = 2$ offers a good balance between exploration and exploitation.
- Pp_t: this parameter underlines the high exploration capacity of AFT when it takes relatively small values. This parameter is gradually increased during the iterative process of AFT. The choice of the values for β_0 and β_1 in Pp_t is a little bit arbitrary, but was selected based on pilot testing for a large set of test functions. In the initial iterations, the candidates are all far away from each other in distance. Updating the parameter Pp_t improves AFT's ability to avoid stagnation entrapment in local optima and approaches the global optimum. Based on empirical testing, $\beta_0 = 0.1$ and $\beta_1 = 2$ present a good balance between exploration and exploitation.
- $\text{sgn}(\text{rand} - 0.5)$: this parameter manages the direction of exploration. Since *rand* takes values between 0 and 1 with a uniform distribution, there is an equal probability of negative and positive signs.
- Marjaneh's intelligence plans: Using this parameter will directly improve the AFT's ability for exploration.

4.5 Exploitation ability of AFT

The key parameters that help to perform local search and exploitation in AFT can be described as follows:

- Td_t: as iteration passes, exploration fades out and exploitation fades in. Small values of Td_t lead to local searches in promising areas of the search space. As a result, at the last iterations, where thieves are close to the house of Ali Baba, the positioning updating process with cases 1, 2 and 3 will assist in local search around the best solution, leading to exploitation.
- Pp_t: this parameter controls the exploitation feature, by quantifying the quantity of exploitation through in-

depth search around the best solution. With the passage of iterations, the exploitation stage heightens with facing relatively large values of this parameter. Thus, the positioning updating process with cases 1 and 2 enhances AFT's ability to locally ramp up the searches in the space, which results in further exploitation.

- $\text{sgn}(\text{rand} - 0.5)$: this parameter controls the quality of exploitation by changing the direction of the search.

4.6 Computational complexity analysis

Computational complexity of an optimization method can be defined by a function that links the runtime of the optimization method to the input size of the problem. To do this, Big-O notation is applied here as a widespread terminology. In this, the time complexity of AFT can be given as follows:

$$\begin{aligned}\mathcal{O}(AFT) =& \mathcal{O}(\text{problem def.}) + \mathcal{O}(\text{initialization}) \\ &+ \mathcal{O}(t(\text{pos. update})) + \mathcal{O}(t(\text{cost function})) \\ &+ \mathcal{O}(t(\text{boundary control}))\end{aligned} \tag{12}$$

As Eq. 12 suggests, the time complexity of AFT relies on the number of iterations (t), the number of thieves (n), the dimension of the problem (d) and the cost of the objective function (c). In concrete terms, the overall time complexity of AFT under the termination method can be computed as follows:

$$\begin{aligned}\mathcal{O}(\text{AFT}) =& \mathcal{O}(1) + \mathcal{O}(nd) + \mathcal{O}(vtnd) \\ &+ \mathcal{O}(vtnc) + \mathcal{O}(vtnd)\end{aligned} \tag{13}$$

where v denotes the number of evaluation experiments.

The number of iterations (t) is typically greater than the number of thieves (n), the cost of the fitness function (c) and the number of problem's variables (d). Also, the number of problem variables (d) and the cost of the fitness function (c) are usually less than the number of thieves (n). Accordingly, the parameters t and n are important factors in assessing the computational complexity. As $nd \ll tnd$ and $nd \ll tcn$, the items 1 and nd can be excluded from the complexity issue given in Eq. 13; also, $2vtnd \cong vtnd$. Therefore, the time complexity of AFT can be expressed as follows:

$$\mathcal{O}(\text{AFT}) \cong (vtnd + vtnc) \tag{14}$$

As it is shown, the complexity issue of the AFT is of the polynomial order, which can be deemed as an effective meta-heuristic optimization algorithm.

4.7 Characteristics of AFT

Human-based algorithms possess two abilities, that is, exploration and exploitation. This is to optimize the search space of a problem. In AFT, these abilities are realized by the convergence of the thieves toward the global optimum solution. To be precise, convergence means that most of the thieves gather in the same position in the search space. AFT utilizes several parameters that lead to exploration and exploitation as explained in Subsects. 4.4 and 4.5, respectively. These parameters are beneficial for carrying out the convergence process of AFT. The thieves (i.e., search agents) in AFT can change their position in line with a mathematical model and tuning criteria as implemented by three basic cases that may occur while thieves search for Ali Baba. These cases are presented in Eqs. 5, 10 and 11. In each case, it is assumed that the thieves search efficiently throughout the surrounding environment, while there is also a percentage that occurs due to Marjaneh's intelligence that forces the thieves to search in random locations. There are two important parameters in AFT, referred to as tracking distance and perception potential that are presented in Eqs. 8 and 9, respectively. With these two parameters, AFT can better search the space for all possible solutions to identify the optimal or suboptimal solutions. Another important parameter in AFT is the simulation of Marjaneh's intelligent ways to deceive the thieves. Thereby, the thieves will explore the search space in different locations and directions, which implies that better solutions may be found in other promising areas. In short, AFT has several distinct merits according its basic principle, summarized as follows: (1) The position updating models for case 1 and case 3 of AFT effectively assist the individuals of the population to explore and exploit every area in the search space. (2) The random search that thieves use in the search space using case 2 not only enhances the diversity of the population but also ensures the speed of convergence, indicating an efficient balance between exploration and exploitation. (3) The number of parameters in AFT is small, but they have good ability to improve its strength and performance. (4) The computational burden of AFT is low as discussed in Subsect. 4.6.

As a result, there is a big room for enhancing the performance of AFT according to the above mentioned characteristics, as presented in the following section.

5 Possible developments of AFT

To further study the potential performance of the AFT algorithm, it is elaborated from several aspects as shown in the following subsections.

5.1 Self-adaptation of tracking distance of AFT

Self-adaptive tracking distance is used to strike a better balance between exploration and exploitation during the search process [70]. This distance decreases as a function of time indicating that the exploration capacity gradually fades out and exploitation capacity gradually fades in. However, the search agents trapped in local optima area demand reasonable exploration to escape from this local optima. Some search agents require a large tracking distance to explore the search space and others exploit the local area with a small tracking distance. So, it is imperative for each search agent to have its own tracking distance to balance exploration and exploitation. When the fitness value of a search agent is worse or unaltered, it denotes that the search agent can identify the local optimal area. In this, the search agents require a large tracking distance to move away from this area. When the fitness value of a search agent ameliorates, it indicates that the search agent has a superior chance of getting close to the optimal solution. Hence, the value of the tracking distance, Td(t), of the search agents of AFT should be grown. Figure 3 presents an illustration of the self-adaptive tracking distance [70].

In Fig. 3, the pentagram, sphere, circle and arrowhead stand for the global optimum area, the local optimum area, search agent and tracking distance, respectively. Case 1 implements that if a search agent is trapped into local optimal area, it needs an appropriate tracking distance to raise its strength to escape from this area. On the other hand, Case 2 indicates that a search agent rapidly moves to the global optimal area with mounting tracking distance. In this context, to estimate the situation of the search agents in AFT, the counters ns_i and nf_i are presented in Eqs. 15 and 16, respectively, to record the fitness results of the ith search agent [70].

$$ns_i^t = \begin{cases} ns_i^{t-1} + 1 & if \;\; f_i(t) < f_i(t-1) \\ 0 & if \;\; f_i(t) \geq f_i(t-1) \end{cases} \tag{15}$$

$$nf_i^t = \begin{cases} nf_i^{t-1} + 1 & if \;\; f_i(t) > f_i(t-1) \\ 0 & if \;\; f_i(t) \leq f_i(t-1) \end{cases} \tag{16}$$

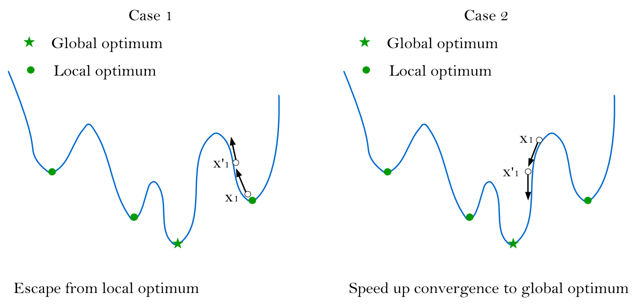

Fig. 3 Illustration of self-adaptive tracking distance in AFT [70]

Equations 15 and 16 are presented to adjust the parameter Td(t) according to the fitness of the objective function, where it is employed to estimate the search condition of search agents. If a search agent fails to obtain a better solution in many iterations, the search agent gets stuck in the local optima area with a high probability. If a search agent is improved in many iterations, it may migrate to the global optima.

A threshold θ and probability p are applied to dominate the update of the tracking distance of search agents over the course of iterations. If ns_i^t exceeds θ, the tracking distance will be increased to speed up the convergence of the ith search agent toward the super search agent. Likewise, if nf_i^t exceeds θ, the tracking distance is enhanced to mend the ability to avert the local optima while searching. The self-adaptive $\mathrm{Td}_i(t)$ of the ith search agent is defined in Eq. 17:

$$\mathrm{Td}_i(t) = \begin{cases} \mathrm{Td}_i(t) \cdot r_i(t) & \text{if count} > \theta \,\&\, \text{rand} < p \\ Td_i(t) & \text{otherwise} \end{cases} \tag{17}$$

where *count* comprises two counters ns_i^t and nf_i^t, and rand is a uniformly distributed random value in the interval [0, 1].

In Eq. 17, when count overrides θ and rand overrides p, the search agent requests a large tracking distance to strengthen its exploration capability, where it is multiplied by $r_i(t)$. Otherwise, the updated tracking distance is set to the tracking distance of AFT.

The tracking distance of search agents is related to the distance that thieves use to follow Ali Baba as given in Eq. 8. Also, there is another parameter that is related to the perception potential of the thieves for Ali Baba. Therefore, the tracking distance and perception potential constants of the ith search agent are used to adapt its $\mathrm{Td}_i(t)$. The ratio of tracking distance and perception potential constants of the search agents can be thought of as the new tracking distance value for the search agents. Here, $r_i(t)$ stands for the adjusted value of $\mathrm{Td}_i(t)$ and is presented in Eq. 18:

$$r_i(t) = \begin{cases} \frac{2}{c} & if \;\; c < 1 \\ c & c \geq 1 \end{cases} \tag{18}$$

where c is the ratio of tracking distance and perception potential constants as shown in Eq. 19.

$$c = \left| log\left(\frac{\mathrm{Td}_i(t)}{\mathrm{Pp}_i(t)}\right) \right| \tag{19}$$

where $\mathrm{Td}_i(t)$ and $\mathrm{Pp}_i(t)$ are the tracking distance and perception potential of search agent x_i at iteration t, respectively.

Equation 18 states that $r_i(t)$ is set to the reciprocal of c when c is less than 1 to ensure that the search agent can have a reasonably large tracking distance, otherwise, $r_i(t)$ is set to c.

The original and self-adaptive tracking distance constants are presented in Figs. 1 and 4, respectively.

The tracking distance constant in Fig. 4 is changed in accordance to the search condition of the search agents, where the red and blue lines indicate a modification of Case 1 and Case 2, respectively. The search agent is in a state of failure when the search agent gets a worse search status. In contrast, the search agent is in a state of success when the search agent gets better search status.

In sum, in the search process, various search agents have different search cases. Some of them want vigorous exploration to explore the solution space, while others want extensive exploitation to locate a better solution. Therefore, each search agent adjusts its Td to balance exploration and exploitation.

5.2 Population hierarchical structure of AFT

The hierarchical structure of the population is that the search agents are arranged and placed in different layers according to some specific characteristics [71, 72]. These layers can be stated as different levels from top to bottom according to the actual effect of each layer on the search agents. The top layer leads its next layer, and its next layer leads its second one, and so on. In this light, the interactive relationship between the layers is created to form a hierarchical structure that is used to guide the evolution direction of the search agents [72]. Here, a hierarchical population structure was used in the AFT algorithm, so that premature convergence could be mitigated and search agents could elicit correct interactive information to realize a better development. Further, the search agents have ample opportunity to escape from local optima and get close to the global optimum. Here, a three-layer hierarchical structure was constructed for hierarchical AFT (HAFT) as follows:

- *Bottom layer* The distribution of all search agents in the current population is displayed on this layer. Search agents move toward better ones in terms of the best search agents on the middle layer. This layer can divulge the landscape of the function created by a big number of search agents.
- *Medium layer* For the effective guidance of the development of general search agents, the best predetermined search agents are ranked on this layer. At each iteration, the medium layer leads the bottom layer to fulfill the position update for the search agents. Each search agent needs a large tracking distance and small perception potential to globally explore the whole search area in the first few iterations of the search process, and these parameters are gradually updated over iterations. This means that the exploration ability of AFT wants to be supported by a large suitable Td(t) and its exploitation capacity demands a small one. Thus, to improve the exploration ability of AFT, a new Td(t) is presented to supersede the original one given in Eq. 8. In the proposed HAFT, a log-sigmoid transfer function was used to design a new constant Td(t) with the formula given below:

$$\mathrm{Td}(t) = \frac{T0}{1 + e^{\frac{t - \frac{L}{2}}{L}}} \tag{20}$$

where L is a step length. It can be observed that the value of Td(t) in the graph shown in Fig. 1 decreases rapidly before 500 iterations, indicating the exploration ability of AFT is rapidly diminishing. On the other side, the value of Td(t) in Fig. 5 consistently preserves a large value before 500 iterations and then drops rapidly to near zero. This effect can ensure a powerful exploration ability of AFT in the early stage so that it has

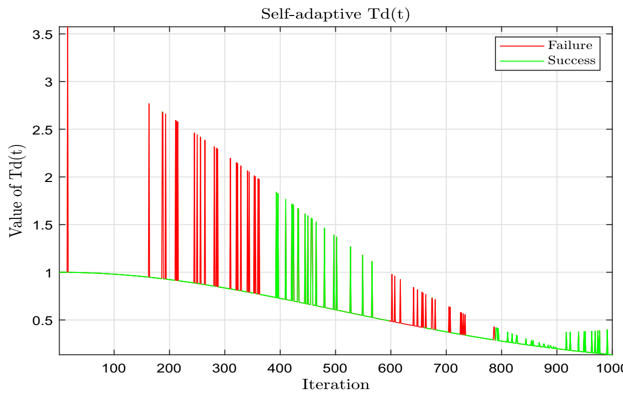

Fig. 4 Proposed exponential function for the self-adaptive tracking distance

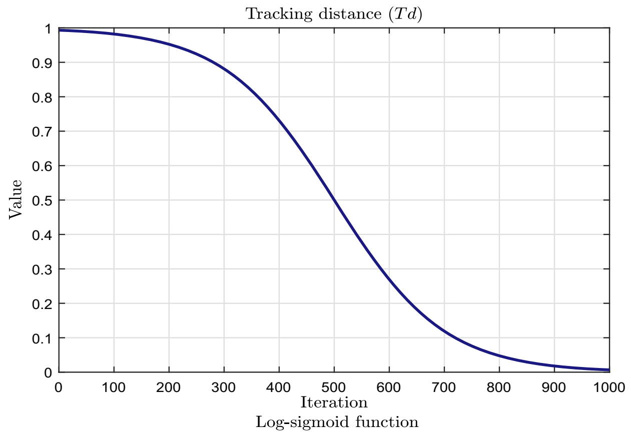

Fig. 5 The new tracking distance parameter Td(t)

enough time to search for an approximate optimal that can be further improved by its next exploitation capability. Since the number of the best search agents gradually decreases with iterations, which means that the number of best search agents dynamically decreases on this layer. It is useful for the global optimum search agent on the top layer to effectively direct many elite search agents in the current population to provide a better development tendency for all of the search agents on the bottom layer.

- *Top layer* To provide efficient management for the middle layer, a global optimum search agent is determined and placed on this layer. At each iteration, the pre-identified best search agents on the medium layer are selected to be compared with the global optimum search agent. In case there is a better search agent, the optimal global one is replaced by it. The global optimal search agent has the best position in the current population. Hence, it can attract many of the best search agents to move toward it. This strategy could prevent the best search agents from being trapped in the local optima and could accelerate the convergence speed of the population. According to this strategy, the formula of updating the position of the best search agents oriented by the global optimal search agent is presented as follows:

$$x^i_{t+1} = g_{opt_t} - \left[\mathrm{Td}_t\left(y^i_t - m^i_t\right)r_2\right]\mathrm{sgn}(0.5 - r) \qquad r_3 < 0.5 \tag{21}$$

where g_{opt} denotes the global optimal search agent and r represents a random number with a uniform distribution in the interval [0, 1].

In HAFT, Eq. 21 was proposed to update the position of the search agents on the top layer which could efficaciously alleviate the early convergence of AFT and improve its performance.

In order to further clarify the apparent properties of HAFT, Fig. 6 was drawn to show its operating precept on the multimodal landscape with local optima.

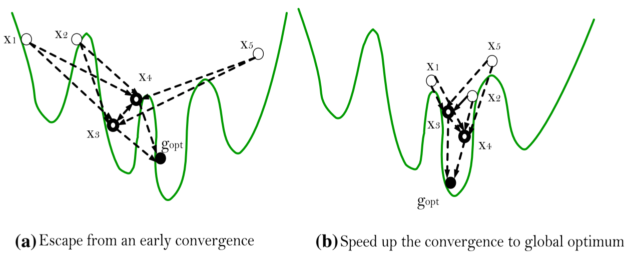
x1
x2
x4
x5
x3
gopt
(a) Escape from an early convergence
x1
x5
x2
x3
x4
gopt
(b) Speed up the convergence to global optimum

Fig. 6 The illustrative diagrams of the operating precept of HAFT [71]

It is obvious from Fig. 6a that search agents x_3 and x_4 are heading toward a local optimal, while the global optimal search agent g_{opt} provides additional guidance to assist them escape from the local optimal. When search agents x_3 and x_4 do not fall into a premature convergence as shown in Fig. 6b, they can draw others to move toward them. Meanwhile, g_{opt} will accelerate the movement of x_3 and x_4 in order to improve the convergence rate of the population. Thus, the capabilities of exploration and exploitation can be enhanced in the search process. More details about hierarchical population structure can be found in [71, 72].

5.3 Exploration and exploitation

Exploration and exploitation are the two most important properties of meta-heuristic algorithms to achieve success when addressing optimization problems [38]. With regard to these two concepts, empirical experiments have shown that there is a robust relationship between the exploration and exploitation ability of a particular search method and its convergence speed. Particularly, while exploitation procedures are known to improve convergence toward the global optimum, they are also known to rise the likelihood of entrapment into local optima [38]. Conversely, search strategies that promote exploration over exploitation incline to increase the likelihood of locating areas within the search space, where the global optimum is more probable to be identified. This is at the cost of deteriorating the convergence speed of optimization algorithms [39]. In recent years, the question of how exploration and exploitation of solutions is realized in meta-heuristics has remained an open subject, and although it appears trivial, it has stayed as a source of contention among many researchers [40]. Although many thoughts and notions may sound opposite, there appears to be a common consent within the research community on the conception that an adequate ratio between exploration and exploitation is necessary to ensure reasonable performance in this type of search methods.

Meta-heuristics use a set of candidate solutions to explore the search area with the goal of finding satisfying solutions for an optimization problem. Generally, the search agents with the superior solutions are liable to guide the search process toward them. As a result of this attraction, the distance between the search agents fades in while the impact of exploitation fades out. On the other side, when the distance between the search agents increases, the influence of exploration strategy is more pronounced. To compute the increase and decrease in distance between the search agents, a diversity measurement [73] is taken into account. Under this method, population diversity is stated as follows [38]:

$$\mathrm{Div}_j = \frac{1}{N}\sum_{i=1}^{N}\left|\mathrm{median}(x^j) - x_i^j\right| \tag{22}$$

$$\mathrm{Div} = \frac{1}{m}\sum_{j=1}^{m}\mathrm{Div}_j \tag{23}$$

where $\mathrm{median}(x^j)$ represents the median of dimension j in the entire population, x_i^j is the dimension j of search agent i, n stands for the number of search agents and m denotes the number of design variables of the optimization problem.

The diversity in each dimension, Div_j, is stated as the distance between the dimension j of each search agent and the median of that dimension on average. The full balance response is defined as the percentage of exploration and exploitation utilized through a given algorithm. These values are calculated at each iteration using the following formulas [38]:

$$\mathrm{XPL}\% = \left(\frac{\mathrm{Div}}{\mathrm{Div}_{\max}}\right) \times 100 \tag{24}$$

$$\mathrm{XPT}\% = \left(\frac{\left|\mathrm{Div} - \mathrm{Div}_{\max}\right|}{\mathrm{Div}_{\max}}\right) \times 100 \tag{25}$$

where $\mathrm{Div}_{\max}$ stands for the maximum diversity value present in the whole optimization process.

The percentage of exploration (XPL%) corresponds to the relationship between the diversity at each iteration and the maximum diversity reached. The percentage of exploitation (XPT%) represents the level of exploitation [38]. As can be observed, both elements XPL% and XPT% are mutually conflicting and complementary. In assessing the balance response, the use of the median value averts discrepancies through the use of a reference element. This balance response is also affected by Div_{max} that is found during the whole optimization process. This value is employed as a reference to assess the rate of exploration and exploitation.

5.4 Chaos in meta-heuristic algorithms

Chaos theory is one of the most effective strategies used to improve the performance of meta-heuristics by fostering their exploration and exploitation features. Chaos appears to exhibit irregular motion, a characteristic often encountered in nonlinear dynamic systems [74, 75]. It appears to be random, unexpected behavior that a deterministic nonlinear system can present under deterministic conditions. Thus, a chaotic system alters randomly and ultimately passes through each state in the search space when the time period is long enough. The applications of chaos in global optimizers fall into two categories.

5.4.1 Chaotic maps and sequences

Chaotic maps are one of the preferable ways to reinforce the performance score of meta-heuristics in terms of both local optima avoidance and convergence property. They are widely used to improve population diversity and solution quality by substituting random values and adjusting parameters in the initialization of population and iterative loop procedures [74, 75]. Chaotic properties have been used in improved and new meta-heuristics, such as EAs [74, 75], the immune system algorithm [76], PSO [77] and DE [78]. These chaotic meta-heuristics have received a high level of performance through the use of chaotic sequences to replace random variables and parameters. In this, they presented superb performance compared to the other corresponding standard meta-heuristics.

5.4.2 Chaotic local search

Chaotic local search (CLS) appears as an applicable option by making use of randomness and ergodicity of chaos [74, 75]. Chaotic search is a mechanism that could be conducted to improve the accuracy of the search and convergence speed. For this reason, CLS has been integrated with several meta-heuristic algorithms and achieved splendid success in enhancing their performance, such as chaotic PSO [79], chaotic DE [80] and chaotic GSA [81]. Their outcomes showed that CLS could prominently strengthen search capacity and dwindle the problem of getting into local optima. It has been widely demonstrated that meta-heuristics with CLS achieved better performance in terms of convergence rate and solution accuracy than the other corresponding original versions [74, 75].

5.5 Theoretical analysis of the AFT algorithm

To theoretically analyze the performance of the AFT algorithm from the perspective of complex network, there is a need to establish a relationship among its search agents. This analysis is helpful and vital to explain its essence and discover some guiding methods to overcome its limitations in order to better foster the performance of this algorithm [82]. For this intent, we use the population interaction network (PIN) method reported in [83, 84] to put in place the relationship between the search agents of AFT. This is for exploring and analyzing the intrinsic phenomenon that occurs in a complex network. A clustering method was used to classify the search agents [83, 84]. In this method, each search agent can be regarded as a vertex and the update position mechanism between the search agents denotes the generation of edges. The PIN method can be used to obtain both the intrinsic connection of knowledge and characteristic of the network formed by the population. The method that composes the interaction of population in AFT is displayed in Fig. 7.

In Fig. 7, blue circles, transparent circles, transparent rectangles, blue rectangles, and blue diamond shapes implement the search agents (i.e., vertices) in the current population, the clusters, search agents x_p that will be replaced, the search agents that have been replaced and the new created search agent U_n, respectively. The circle, square and triangle denote the current, new constructed and old substituted vertices, respectively. It is noticed from Fig. 7 that the distribution of search agents has changed in the whole population and that the number of search agents in each class changes accordingly. In sum, the initial construction process of PIN can be described as stated below:

1. There are three classes and nine basic search agents in the population;
2. Two chosen search agents, x_{s1} and x_{s2}, from two classes yield the new created search agent U_n for comparison with the previous search agent, x_p. A vertex and two edges are created at the same time;
3. If the search agent U_n overrides the search agent x_p, in which $(f(U_n) < f(x_p))$, then U_n replaces x_p;
4. Another search agent x_s is chosen from one class to create U_n to replace x_p, which means creating a vertex and an edge;
5. The replacement process is carried out once more when $f(U_n) < f(x_p)$;
6. At the next iteration, the clustering method resumes classifying the search agents into three classes to finally get to terminate the algorithm and obtain the PIN topology.

Readers can read [83, 84] for a detailed description of the PIN method.

6 Comparative analysis of AFT with other meta-heuristics

This section presents a comparative analysis of AFT with other meta-heuristics such as PSO, GSA, DE, GA, covariance matrix adaptation-evolution strategy (CMA-ES) and ant colony optimization (ACO) algorithm.

6.1 Particle swarm optimization

PSO [85] mimics the cooperative social collective behavior of the living creatures such as flocks of birds. Optimization begins with the use of randomly generated solutions known as artificial particles. Each particle in the swarm has a randomly generated velocity. If x_i is the initial position of the ith particle with velocity v_i, then the position updating strategy of PSO can be given as follows [86]:

$$v_i(t+1) = wv_i(t) + c_1(Pbest_i - x_i(t))r_1 + c_2(Gbest - x_i(t))r_2 \tag{26}$$

$$x_i(t+1) = x_i(t) + v_i(t+1) \tag{27}$$

where w is the inertial weight, c_1 and c_2 are cognitive and social constants, respectively, r_1 and r_2 are distributed random numbers in the interval [0, 1], $Pbest_i$ is the local best solution of the ith particle and $Gbest$ is the global best solution among all particles.

6.1.1 AFT versus PSO

Similar to PSO, AFT initiates the optimization process by motivating the search agents to move in the search space in search of their target. However, the positioning updating mechanism of AFT is entirely different from that of PSO. Some of the main differences are described as follows:

1. In PSO, the movement update of the ith particle is obtained by $Pbest_i$ and $Gbest$ as given in Eq. 26, where the effect of these two parameters is considered to identify the new position of the particles in the search space. In regard to AFT, the new position of the search agents are obtained through three different cases as given in Eqs. 5, 10 and 11. In other words, PSO updates all solutions with one strategy as presented in Eq. 27, while the search agents of AFT use three strategies to update their position in the search space.
2. The PSO algorithm is greatly influenced by the initial values of the cognitive and social parameters as well as the weighting strategy of the velocity vector, where these parameters are used as the particle develops a new position. In the AFT model, the thieves develop a new position with the help of tracking distance which

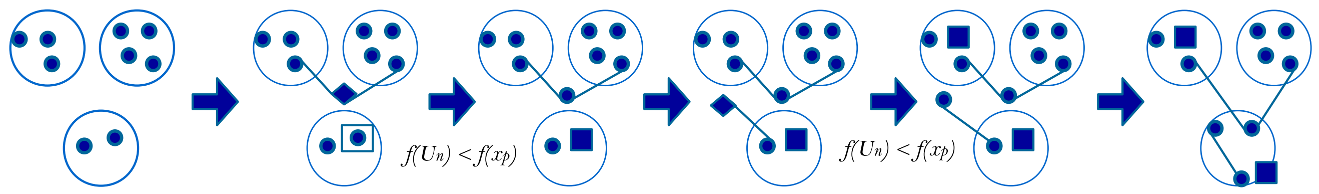

Fig. 7 A descriptive schematic diagram of PIN method in the AFT algorithm [83, 84]

is adapted during its iterative loops. This enables AFT to alternate between local and global searches.

3. The behavior of the thieves' movement is affected by the information given by someone about the whereabouts of Ali Baba's house, which is designed with a random number (r_3). Accordingly, case 1 or case 3 shown in Eqs. 5 or 11 are used, respectively. Inclusion of this random number in the AFT model suddenly redirects thieves' movement and thus improves exploration and exploitation in AFT. On the other hand, PSO does not use such behavior.
4. Simulation of thieves' behavior in Eq. 10 imparts an opportunity to present a random behavior of thieves' movement. This enables AFT to mitigate stumbling in local optimum areas. This behavior is not used in PSO due to the natural behavior of swarms.
5. The use of Marjaneh's intelligence that is formulated in Eq. 7 improves the exploration feature of AFT, where there is no such thing in the PSO algorithm.
6. The use of tracking distance and perception probability in AFT enables it to conduct local searches in local areas at some times, exploration of the search space on a global scale at other times as well as getting an appropriate balance between exploration and exploitation features. These two parameters are not present in the PSO algorithm.

6.2 Gravitational search algorithm

Gravitational search algorithm is a physics-based algorithm evolved on the basis of the law of gravity [44]. Each individual (i.e., agent) evolves its position according to the gravitational force among individuals. The mechanism of GSA is based on the interaction of masses in the universe by means of the Newtonian law of gravitation. To describe GSA, consider a system with N masses (i.e., agents), where the position of the ith mass is defined as follows:

$$X_i = (x_i^1, x_i^2, \ldots, x_i^d) \qquad i \in 1,2,3,\ldots,N \tag{28}$$

where x_i^d denotes the position of the ith mass in the dth dimension and d represents the total number of dimensions in the search space.

The mass of the ith agent is computed after calculating the fitness of the current populations, which is defined as follows:

$$m_i(t) = \frac{f_i(t) - \text{worst}(t)}{\text{best}(t) - \text{worst}(t)} \tag{29}$$

$$M_i(t) = \frac{m_i(t)}{\sum_{j=1}^{N} m_j(t)} \tag{30}$$

where $M_i(t)$ and $f_i(t)$ represent the mass and fitness values of the ith agent at iteration t, respectively, best(t) and worst(t) represent the best and worst fitness values of the current population in the tth iteration, respectively, where worst(t), for a minimization problem, is defined as follows:

$$\text{worst}(t) = \max fit_j(t), \qquad j \in \{1,2,\ldots,N\} \tag{31}$$

The gravitational force between agents X_i and X_j in the dth dimension can be computed as follows:

$$F_{ij}^d(t) = G(t)\frac{M_i(t) \times M_j(t)}{R_{ij}(t) + \epsilon}\left(x_j^d(t) - x_i^d(t)\right) \tag{32}$$

where $R_{ij}(t)$ stands for the Euclidean distance between agents i and j, ϵ is a small value used to eschew division by zero and $G(t)$ is a gravitational constant given as a function of time as shown below:

$$G(t) = G_0 \times e^{-\alpha\frac{t}{T}} \tag{33}$$

where G_0 represents an initial value, α represents a constant value, and t and T represent the current iteration number and the maximum number of iterations, respectively. The total gravitational force $F_i^d(t)$ for agent X_i is given as follows:

$$F_i^d(t) = \sum_{j \in Kbest, j \neq i} \text{rand}_i F_{ij}^d(t) \tag{34}$$

where $Kbest$ refers to a group of the first K best agents with the best fitness value and biggest mass is $kbest$, K is the agent number of $Kbest$ and rand$_i$ is a uniformly distributed random number in the range [0, 1].

Hence, the acceleration $a_i^d(t)$ of agent X_i in the dth dimension at time t can be computed using a law of motion as shown in Eq. 35.

$$a_i^d(t) = \frac{F_i^d(t)}{M_i(t)} \tag{35}$$

Then, the velocity $v_i^d(t+1)$ and position $x_i^d(t+1)$ of agent X_i are updated, respectively, as follows:

$$v_i^d(t+1) = \text{rand}_i \times v_i^d(t) + a_i^d(t) \tag{36}$$

$$x_i^d(t+1) = x_i^d(t) + v_i^d(t+1) \tag{37}$$

where rand$_i$ is a uniformly distributed random value in the interval from 0 to 1.

6.2.1 AFT versus GSA

1. As shown in Eqs. 5, 10 and 11, the position updating mechanism of the search agents of AFT is totally different from that of GSA as defined in Eq. 37.
2. GSA uses acceleration and velocity vectors for the movement of its agents, while AFT generates new

directions of movement for its search agent by various mechanisms.

3. There are some own adaptive parameters for AFT such as Td_t and Pp_t. However, GSA does not use these parameters, where it has its own parameters such as ϵ and $R_{ij}(t)$.
4. AFT algorithm incorporates the concept of random movement of the thieves based on the parameter Pp_t, and uses Marjaneh's intelligence in its position updating mechanism. Obviously, the generation of new solutions by AFT seems very different from the update mechanism of the agents of GSA.

6.3 Conventional differential evolution algorithm

The differential evolution (DE) algorithm is a population-based evolutionary algorithm evolved to solve real-valued optimization problems [87]. The evolutionary process of DE involves evolutionary concepts such as mutation, crossover and selection strategies similar to those used by GAs. The initialization of each individual $X_i, i \in \{1, 2, \ldots, \mathrm{NP}\}$ in DE is described as follow:

$$X_i^d = X_i^{dl} + \mathrm{rand}(0,1) \cdot (X_i^u - X_i^l) \tag{38}$$

where NP is the population size, $d \in \{1, 2, \ldots, D\}$ denotes the dimension of the problem, u and l represent the upper and lower bounds of X_i in the dth dimension, respectively.

The mutation strategy of DE can characteristically create a mutant vector to be an intermediate variable V_i for evolution according to:

$$V_i = X_{r1} + F \cdot (X_{r2} - X_{r3}) \tag{39}$$

where r_1, r_2 and $r_3 \in \{1, 2, \ldots, \mathrm{NP}\}$ are random indices, $i \neq r_1 \neq r_2 \neq r_3$ and F is a constant operator that indicates the level of amplification.

The crossover strategy of DE that can boost the diversity of new agent U_i by combining the original agent X_i with the intermediate variable V_i can be defined as follows:

$$U_i^d = \begin{cases} V_i^d & \text{if } \mathrm{rand}(0,1) \leq \mathrm{CR} \text{ or } d = d_{\mathrm{rand}} \\ X_i^d & \text{otherwise} \end{cases} \tag{40}$$

where CR represents a crossover control parameter and $d_{\mathrm{rand}} \in \{[1, 2, \ldots, D]\}$ denotes a random number.

The selection process in DE is performed in each iteration by contrasting U_i with X_i using a greedy norm for a better agent reserve in the population for the next iteration. Through these evolutionary processes, DE could rapidly converge and eventually obtain the global optimum.

6.3.1 AFT versus DE

Generally speaking, since AFT is a human-based optimization algorithm, so there is no need for evolutionary processes such as crossover, mutation and selection operations. The main differences between DE and AFT can be briefed by the following points:

1. The AFT algorithm preserves search space information over subsequent iterations, while the DE algorithm discards the information of previous generations once a new population is formed.
2. AFT involves fewer operators to adjust and run as compared to DE that uses several operations such as selection and crossover. Moreover, AFT utilizes a parameter denoting Marjaneh's plans, while DE does not memorize the best solution obtained so far.
3. In DE, exploration is enhanced using crossover and selection operations, while in AFT, it is enhanced by allowing the thieves to randomly explore the search space.
4. In DE, mutation is generally implemented on the basis of enhancing the exploitation of DE. However, a better exploitation of AFT is achieved with the perception probability parameter.

6.4 Genetic algorithm

GA was first put forwarded by Holland [88]. It is considered as a global optimization algorithm inspired by biological mechanisms such as evolution and genetics. When using GAs, the search space is used to construct chromosomes, whereby every possible solution is coded as a chromosome (i.e., individual). In optimization with GA, evolution begins with a group of randomly formed individuals from a population. The fitness score of each individual is computed in each generation. The variables of the solutions are adjusted based on their fitness values. Since the best individuals are given higher probability to participate in enhancing other solutions, the random initial solutions are very probable to be improved. Based on a fitness function, chromosomes are selected and certain genetic operators such as mutation and crossover are applied to the selected chromosomes to form new ones. The idea is that these chromosomes evolve and always create better individuals until they reach the global optimum [89].

6.4.1 AFT versus GAs

Both GAs and AFT are population-based techniques; however, the key differences between them can be briefed as follows:

1. Since GA is similar to DE in which it uses crossover, mutation and selection operations; the AFT algorithm does not use these operations.
2. The AFT algorithm uses Marjaneh's intelligence, while GA does not use such a parameter and does not save the best solutions obtained so far.
3. GA evolves and updates its population using crossover, mutation and selection operations, while AFT improves its exploration ability with the concept of random relocation of the thieves that is managed by a perception probability parameter.

6.5 Covariance matrix adaptation-evolution strategy

The CMA-ES [90, 91] is an evolutionary algorithm for nonlinear non-convex optimization problems in continuous domain. Specifically, it is a second-order approximation algorithm that estimates the derivatives of a covariance matrix within an iterative procedure according to the principle of maximum likelihood. By doing this, it tends to maximize the probability of the distribution. At each iteration, the members of the new population are sampled from a multivariate normal distribution N with covariance $C \in \mathbb{R}^{n\times n}$ and mean $m \in \mathbb{R}^n$. The new individuals at generation $i+1$ are sampled as:

$$x_k^{i+1} \sim m^i + \sigma^i \mathcal{N}\left(0, C^i\right) \qquad k = 1, \ldots, \lambda \tag{41}$$

where σ^i is the ith step size and x_k^i is the k_{th} individual at generation i.

The sampled points, λ, are ranked in ascending order of fitness, and the best points, μ, are chosen. The mean of the sampling distribution given in Eq. 41 is updated using *weighted intermediate recombination* of these specified points:

$$m^{i+1} = \sum_{j=1}^{\mu} \omega_j X_{j:\lambda}^{i+1} \tag{42}$$

with

$$\sum_{j=1}^{\mu} \omega_j = 1, \qquad \omega_1 \geq \omega_2 \geq \ldots \geq w_\mu > 0 \tag{43}$$

where ω_j are positive weights, and $x_{j:\lambda}^{i+1}$ stands for the jth ranked individual of the λ sampling points x_k^{i+1}. The sample weights of the standard CMA-ES implementation are decreased as:

$$\omega_j = log\left(\frac{\lambda - 1}{2} + 1\right) - log(j) \tag{44}$$

The covariance matrix can be adapted for the next generation using a combination of rank-μ and rank-one update as follows:

$$\begin{aligned} C^{i+1} = &(1 - c_{cov})C^i + \frac{c_{cov}}{\mu_{cov}} p_c^{i+1} p_c^{(i+1)^T} \\ &+ c_{cov}\left(1 - \frac{1}{\mu_{cov}}\right) \sum_{j=1}^{\mu} \omega_j y_{j:\lambda}^{i+1} (y_{j:\lambda}^{i+1})^T \end{aligned} \tag{45}$$

With $\mu_{cov} \geq 1$ is the weighting between rank-one update and rank-μ, $c_{cov} \in [0, 1]$ is the learning rate for the covariance matrix update, and $y_{j:\lambda}^{i+1} = (X_{j:\lambda}^{i+1} - m^i)/\sigma^i$. The evolution path p_c^{i+1} and σ^i are identified by an adaptation formula [91].

6.5.1 AFT versus CMA-ES

Both CMA-ES and AFT are population-based techniques; however, the major differences between them are summarized as shown below:

1. The CMA-ES basically parameterizes the multivariate normal distribution $\mathcal{N}(textbfm, \sigma^2 C)$ which consists of three terms: the mean vector $\mathbf{m}$, the step-size σ and the covariance matrix C. On the other hand, the AFT algorithm does not use these components, rather it uses Eqs. 5, 10 and 11 to update the position of its search agents.
2. The CMA-ES uses two evolution paths that accumulate consecutive steps of the mean vector update for the cumulative step-size adaptation and the rank-one update of the covariance matrix. However, the AFT algorithm does not use such these evolution paths.
3. AFT uses some concepts to assist alternating between local and global solutions in the update of its search agents' position. However, the CMA-ES algorithm uses a covariance matrix that can be adapted for the next generation using an integration of rank-μ and rank-one.

6.6 Ant colony optimization

ACO is a meta-heuristic algorithm that distributes the search activities to so-called "ants" [92]. The activities are split among agents with simple basic abilities that imitate, to some extent, the behavior of real ants in foraging. It is crucial to underline that ACO has not been developed as a simulation of ant colonies, but to employ the metaphor of artificial ant colonies and their application as an optimization tool. At the start of processing in ACO, where there is no information about the path to go from one point to another, the choice of ants about which path to walk in is totally random. During processing, the intention is that if an ant has to choose between different paths at a given point, those that have been chosen heavily by the preceding ants (i.e., those with a high trail level) are chosen with a higher probability. Generally, ACO approach tries to address an optimization problem by repeating the next two steps:

– A pheromone model, as a specific probability distribution, was used to evolve the candidate solutions over the solution space;
– The candidate solutions are utilized to adjust the pheromone values in a manner that is deemed to bias future sampling toward higher quality solutions.

The choice of ant agents in constructing solution components using a pheromone model is probabilistically defined at each construction step. An ant moves from node i to node j using the following rule:

$$\rho_{(i,j)} = \frac{\tau^{\alpha}_{(i,j)} \cdot n^{\beta}_{(i,j)}}{\sum(\tau^{\alpha}_{(i,j)}) \cdot (n^{\beta}_{(i,j)})} \tag{46}$$

where $\tau^{\alpha}_{(i,j)}$ is the pheromone value associated with edge (i, j), $n^{\beta}_{(i,j)}$ is the heuristic value associated with edge (i, j), α is a positive real parameter whose value identifies the relative importance of pheromone versus heuristic information and controls the influence of $\tau^{\alpha}_{(i,j)}$, β is a positive parameter that identifies the relative importance of pheromone versus heuristic information and controls the influence of $n^{\beta}_{(i,j)}$.

Once the solution is built, the ant evaluates the partial solution to be used using Eq. 47 that specifies how much pheromone to deposit.

$$\tau_{(i,j)} = (1-\rho)\tau_{(i,j)} + \delta\tau_{(i,j)} \tag{47}$$

where $\tau_{(i,j)}$ is the pheromone value correlated with edge (i, j), $\rho \in (0, 1]$ is the pheromone evaporation rate and $\delta\tau_{(i,j)}$ is the amount of pheromone deposited, typically given by:

$$\tau^{k}_{(i,j)} = \begin{cases} 1/L_K & \text{if ant } K \text{ travels on edge } (i,j) \\ 0 & \text{otherwise} \end{cases} \tag{48}$$

where L_K is the cost of the Kth ant's tour.

6.6.1 AFT versus ACO

ACO and AFT apparently look similar but are quite different. They both present several differences in their formulation and position updating mechanism.

1. Both ACO and AFT work on the effective division in the search for the optimal solution. In ACO, the idea is to create a pool of artificial ants that move randomly around an environment. In AFT, the thieves search for Ali Baba by exploring and exploiting each area in the search space using three different cases in the position updating process.
2. In ACO, the candidate solutions are constructed using a pheromone model. However, AFT uses the local and global best solutions of the thieves to locate the optimum solutions.
3. In ACO, a new solution is created by Eq. 48, which is conceptually different from the position updating strategy of AFT given in Eqs. 5 to 11. The updating strategy of AFT is a kind of directed and undirected search approach, in which new solutions are forced to move toward a better solution.
4. The AFT algorithm uses a memory parameter (i.e., Marjaneh's plans) in its updating process. On the other hand, ACO does not use such a parameter in updating its new solutions. Apart from this, AFT also uses a stochastic location updating strategy as shown in Eq. 10 to improve its exploration feature, while ACO does not use such a strategy.
5. AFT has two parameters that can be adapted during its iterative process to enhance exploration and exploitation features and to balance them. However, ACO does not use any parameters to be adapted over the course of iterations.

As previously discussed, an effective meta-heuristic must strike an appropriate balance between exploration and exploitation. However, there is no rule of thumb [93] to make this happen. The slight differences in solutions update and random distributions could have a significant effect on the performance of the designed algorithms [94]. Therefore, AFT becomes a good competitor to the existing meta-heuristics.

7 Experimental results and analysis

In this section, to assess the accuracy of the proposed AFT algorithm, we conducted intensive evaluations on a set of 62 test functions, involving commonly used unimodal, multimodal, hybrid and composition functions. These functions involve: (1) 23 benchmark test functions (described in Table 33 in "Appendix A"), (2) 29 functions taken from IEEE CEC-2017 benchmark functions [95] described in Table 34 in "Appendix B", and (3) a set of 10 IEEE CEC-2019 functions (described in Table 35 in "Appendix C"). These functions are over and over used in the literature to test the performance of any new meta-heuristic algorithm. The experiments designed to verify the performance of AFT are outlined as follows:

- First, exhaustive comparative studies were presented to verify the reliability and accuracy of the AFT algorithm in relation to other meta-heuristics.
- Second, a set of qualitative measures including search history, trajectory, convergence curves and average fitness values were plotted to examine the adequacy of AFT in addressing several types of test functions.
- Third, the optimization performance of AFT was studied in light of several development applied to AFT from several aspects.
- Forth, Friedman's and Holm's test methods were used to verify the significance of the outcomes obtained by AFT.

7.1 Experimental setup

The results produced by AFT in the optimization of the above three mentioned benchmark functions are compared with those produced by other well-regarded algorithms. The parameter definitions of AFT algorithm and those comparative algorithms are given in Table 1.

In order to provide a fair comparison between the proposed algorithm and another selected set of algorithms, we followed the same initialization process for all the compared algorithms. For all experiments, the common parameter settings for all algorithms are set as follows: The number of individuals used in the search process is set to 30, the number of iterations is set to 1000, and the maximum number of function evaluations (NFEs) for all benchmark functions is set to $d \times 10^3$, where d represents the dimension of the test functions. For each function, the executed runs for each algorithm are repeated 30 times independently to obtain the statistical results. The stop condition for all algorithms was set to the maximum number of iterations. The average fitness (Ave) and standard deviation (Std) values were computed over thirty independent runs to explore the accuracy and stability of the proposed algorithm compared to others. The best scores are shown in bold throughout this paper.

Table 1 Parameter setting values of the AFT algorithm and other algorithms

Algorithm	Parameter	Value
All algorithms	Population size	30
	Number of iterations	1000
AFT	α_0, α_1	1.0, 2.0
	β_0, β_1	0.1, 2.0
MFO [96]	Convergence constant	[− 1, − 2]
	Logarithmic spiral	0.75
SOA [97]	Control parameter (A)	[2, 0]
	f_c	2
CSA [98]	flight length (*fl*)	2.0
	Awareness probability (*AP*)	0.1
SHO [99]	Control parameter (**h**)	[5, 0]
	M Constant	[0.5, 1]
GWO [100]	Control parameter (**a**)	[2, 0]
MVO [45]	Wormhole Existence Prob.	[0.2, 1]
	Traveling distance rate	[0.6, 1]
SCA [101]	Number of elites	2
PSO [85]	c_1, c_2	1.8, 2
	w	[0.9, 0.1]
GSA [44]	Alpha, G_0	20, 100
	Rnorm, Rpower	2, 1
GA [102]	Crossover	0.9
	Mutation	0.05
DE [87]	Crossover	0.9
	Scale factor (F)	0.5
SSA [11]	Control parameter (c_1)	0.5
DA [57]	Coefficient (**s**)	[0.2, 0]
	Coefficient (**e**)	[0.1, 0]
	Coefficients (**a**, **c**, **f**)	[0, 0.2]
WOA [103]	Control vector (**a**)	[2, 0]
ACO [92]	β, ρ, α	[2, 0.5, 0.3
CMA-ES [90]	λ, μ	$4 + 3ln(n)$,$\lambda/2$
	$w_{i=1,\dots,\mu}$	$ln\left(\frac{\lambda+1}{2}\right) - ln(i)$
	c_c, c_{cov}	$4/(n+4)$, $\frac{2}{(n+\sqrt{2})^2}$
	c_σ, d_σ	$4/(n+4)$, $c_\sigma^{-1} + 1$

7.2 Classical benchmark test functions

Twenty-three widely used benchmark test functions were used to evaluate the overall performance of AFT and to compare it to other optimization algorithms. These test functions were carried out under minimization problems which can be categorized into three main kinds: unimodal [99], multimodal [96] and fixed-dimension multimodal [99]. "Appendix A" presents the mathematical description of these categories. The characteristics of the unimodal (F_1–F_7), multimodal (F_8–F_{13}) and fixed-dimension multimodal (F_{14}–F_{23}) functions are detailed in Table 33. These three groups of test functions are widely accepted in the literature as benchmark evaluation functions due to:

- The functions in the first group (i.e., unimodal functions) have only one global optimum with no local optimum. This group is usually used to assess the convergence behavior as well as the exploitation power of any new or enhanced optimization algorithm.
- The functions in the second and third sets (i.e., multimodal and fixed-dimension multimodal functions) have several local optimum and more than one global

optimum. These functions are effective in examining the ability of AFT to avoid the local optimums and in evaluating its exploration feature.

The average processing times elapsed by AFT and other comparative algorithms in optimizing the classical benchmark test functions are given in Table 2 and graphically illustrated in Fig. 8.

It can be seen that AFT outperforms other algorithms as it takes less processing time than other algorithms. Therefore, it can be deduced that the computational efficacy of AFT is much better than other competitors.

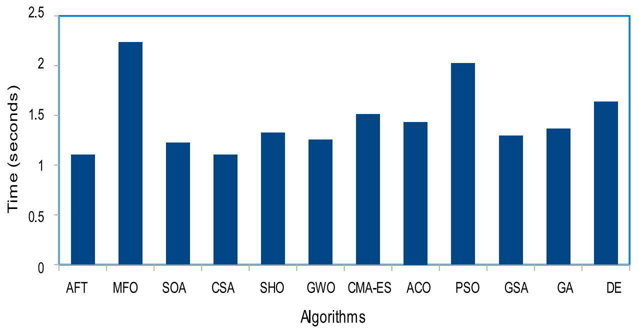

Fig. 8 Average running time of the proposed algorithm and other optimization algorithms

7.2.1 Performance of AFT in unimodal functions

As discussed above, the unimodal functions (F_1–F_7) are beneficial to assess the exploitation capability of optimization algorithms. The average fitness and standard deviation results of AFT and other algorithms are displayed in Table 3. These values are recorded after running the experiments 30 times for each algorithm.

It can be seen from Table 3 that the performance of AFT is very efficient compared to other competitors. In particular, the AFT algorithm scored the best scores for functions F_4, F_5 and F_6. Also, it achieved the best score equally with SHO algorithm for functions F_1, F_2 and F_3. From an engineering point of view, it does not matter whether the algorithm finds the optimal result such as 10E-05 or 10E-25; both are considered zero. It is clear that the SHO algorithm is a competitive algorithm such that it got the best scores for functions F_1 and F_3. Although SHO got the best results for functions F_1 and F_3, the difference between it and AFT is of a very small value. The SOA algorithm obtained the best results for function F_7. Once again, the proposed AFT algorithm got the best standard deviation values for functions F_4–F_6. These outcomes clearly indicate that the AFT algorithm is robust, reliable and highly effective compared to other widely used algorithms in the literature. The AFT algorithm achieved either the best score or competitive results in functions F_1–F_7. The very small values of STD of the AFT algorithm reveal the high level of stability that this algorithm has.

Table 2 Average running time of AFT and other meta-heuristic algorithms

Algorithm	Average time (in seconds)
Ali Baba and the forty thieves	1.1043
Moth flame optimization	2.2362
Seagull optimization algorithm	1.2245
Crow search algorithm	1.1045
Spotted hyena optimizer	1.3269
Grey wolf optimizer	1.2536
Covariance matrix adaptation-ES	1.5114
Ant colony optimization	1.4332
Particle swarm optimization	2.0241
Gravitational search algorithm	1.2965
Genetic algorithm	1.3668
Differential evolution	1.6354

7.2.2 Performance of AFT in multimodal functions

In order to evaluate the capabilities of optimization algorithms for local optimum avoidance and exploration potency, the multimodal (F_8–F_{13}) and fixed-dimension multimodal (F_{14}-F_{23}–) are employed in the literature as benchmark functions for evaluating algorithms for this purpose. The outcomes are given in Tables 4 and 5 to show the Ave and Std, over 30 independent runs, for all the compared algorithms in multimodal and fixed-dimension multimodal functions, respectively.

The results shown in Table 4 point out that the performance of AFT is also very efficient when employed to solve multimodal problems. Particularly, the AFT algorithm scored the best scores for functions F_8, F_{10}, F_{12} and F_{13}. GSA got the second best score for function F_{13}, and SHO got the best scores for functions F_9 and F_{11}, where the results of SHO in these functions are nearly close to the results released by the AFT algorithm. These outcomes once again confirm the reliability and stability of AFT since it has very small values for *Std*.

The results given in Table 5 confirm the superiority of AFT, which gained the best mean fitness values, either individually or equally with other algorithms. The CSA was competitive and got the best average fitness equally with the proposed AFT algorithm in F_{14}, F_{16}, F_{17} and F_{18} test functions. Although the AFT algorithm did not achieve the best average fitness values for F_{19} and F_{20}, its results are very comparable to those algorithms that obtained the

Table 3 Results of the AFT algorithm and other meta-heuristics in unimodal benchmark test functions

F	AFT		MFO		SOA		CSA		SHO		GWO	
	Ave	Std	Ave	td	Ave	Std	Ave	Std	Ave	Std	Ave	Std
F_1	9.52E-69	2.80E-68	3.15E-04	5.00E-05	**0.00E+00**	**0.00E+00**	1.98E-08	2.52E-08	**0.00E+00**	**0.00E+00**	4.69E-47	4.50E-48
F_2	3.5E-30	1.36E-29	3.71E+01	2.16E+01	**0.00E+00**	**0.00E+00**	2.12E-03	2.64E-03	**0.00E+00**	**0.00E+00**	1.20E-24	1.00E-24
F_3	2.70E-42	7.67E-42	4.42E+03	3.71E+03	4.62E-19	2.30E-27	3.36E-04	5.80E-04	**0.00E+00**	**0.00E+00**	1.00E-14	4.18E-15
F_4	**3.36E-23**	**9.43E-23**	6.70E+01	1.06E+01	7.35E-05	3.11E-06	1.36E-03	2.42E-03	7.78E-12	5.06E-12	2.02E-14	2.95E-15
F_5	**7.97E-01**	**1.12E-01**	3.50E+03	2.83E+03	7.00	1.13E-01	1.70E+01	3.08E+01	8.59	5.53E-01	2.79E+01	1.84
F_6	**0.00E+00**	**0.00E+00**	1.66E-04	2.49E-05	3.47E-02	1.31E-04	2.89E-08	2.57E-08	2.46E-01	1.78E-01	6.58E-01	3.38E-01
F_7	4.81E-01	2.60E-01	3.22E-01	2.93E-01	**3.35E-06**	**7.51E-06**	4.66E-01	2.80E-01	3.29E-05	2.43E-05	7.80E-04	3.85E-04
F	CMA-ES		ACO		PSO		GSA		GA		DE	
	Ave	Std	Ave	Std	Ave	Std	Ave	Std	Ave	Std	Ave	Std
F_1	1.42E-01	3.13E-18	2.97E-18	2.12E-18	4.98E-09	1.41E-10	1.16E-16	6.10E-17	1.95E-12	2.04E-13	3.10E-10	3.21E-11
F_2	2.98E-07	1.78	1.42E-15	4.41E-16	7.29E-04	1.57E-05	1.70E-01	5.29E-02	6.53E-18	4.00E-19	4.99E-13	4.81E-14
F_3	1.59E-05	2.21E-05	2.40E-03	1.09E-02	1.40E+01	7.13	4.16E+02	1.56E+02	7.70E-10	5.46E-12	5.58E-02	1.50E-03
F_4	2.01E-06	1.25E-06	9.80E-9	1.17E-9	6.00E-01	1.72E-01	1.12	9.89E-01	9.17E+01	5.67E+01	5.21E+01	1.01E-01
F_5	36.79	33.46	17.08	0.16	4.93E+01	3.89E+01	3.85E+01	3.47E+01	5.57E+02	4.16E+01	2.10E+02	1.73E+02
F_6	6.83E-19	6.71E-19	3.29E-18	5.19E-19	9.23E-09	8.48E-10	1.08E-16	4.00E-17	3.15E-01	9.98E-02	9.77E-02	5.13E-02
F_7	2.75E-02	0.79E-02	0.39E-02	0.13E-02	6.92E-02	2.87E-02	7.68E-01	5.70E-01	6.79E-04	5.21E-04	4.00E-03	2.27E-03

The best results are written in bold for the purposes of confirming the best algorithm

Table 4 Results of the AFT algorithm and other meta-heuristic algorithms in multimodal benchmark test functions

F	AFT		MFO		SOA		CSA		SHO		GWO	
	Ave	Std	Ave	Std	Ave	Std	Ave	Std	Ave	Std	Ave	Std
F_8	**− 1.11E+03**	**2.69E+02**	− 8.04E+03	8.80E+02	− 8.50E+03	4.57E+02	− 2.72E+03	3.01E+02	− 1.16E+03	2.72E+02	− 6.14E+03	9.32E+02
F_9	7.70	4.72	1.63E+02	3.74E+01	3.12E+02	2.85E+0	2.30E+01	1.46E+01	**0.00E+00**	**0.00E+00**	4.34E-01	2.68E-01
F_{10}	**5.98E-15**	**2.01E-15**	1.60E+01	6.18	4.22E-16	4.09E-13	1.61	9.31E-01	2.48	1.41	1.63E-14	3.14E-15
F_{11}	1.76E-01	1.11E-01	5.03E-02	1.29E-02	0.00	0.00	1.48E-01	7.89E-02	**0.00E+00**	**0.00E+00**	2.29E-03	1.21E-03
F_{12}	**3.91E-17**	**2.14E-16**	1.26	1.83	5.80E-01	4.71E-02	3.70E-01	6.43E-01	3.68E-02	1.15E-02	3.93E-02	2.42E-02
F_{13}	**1.34E-32**	**5.56E-48**	7.24E-01	1.59E-01	8.48E-02	4.32E-04	4.30E-03	7.18E-03	9.29E-01	9.52E-02	4.75E-01	2.38E-01
F	CMA-ES		ACO		PSO		GSA		GA		DE	
	Ave	Std	Ave	Std	Ave	Std	Ave	Std	Ave	Std	Ave	Std
F_8	− 7007.10	773.94	1.36E+04	61.19	6.01E+03	1.30E+03	2.75E+03	5.72E+02	5.11E+03	4.37E+02	3.94E+03	5.81E+02
F_9	25.33	8.55	15.34	9.56	4.72E+01	1.03E+01	3.35E+01	1.19E+01	1.23E+01	1.01E+01	8.50E+01	1.00E+01
F_{10}	15.58	7.92	1.17E-09	5.12E-09	3.86E-02	2.91E-0	8.25E-09	1.90E-09	5.31E-11	9.10E-12	7.40E-06	5.44E-07
F_{11}	5.76E-15	6.18E-15	4.11E-04	5.12E-05	5.50E-03	4.32E-04	8.19	3.70	3.31E-06	1.43E-06	7.09E-04	5.58E-04
F_{12}	2.87E-16	5.64E-16	1.72E-15	1.19E-15	1.05E-10	2.86E-11	2.65E-01	1.13E-01	9.16E-08	2.80E-08	1.06E-06	1.00E-06
F_{13}	3.66E-04	0.2E-02	2.27E-09	2.28E-09	4.03E-03	3.01E-03	5.73E-32	1.30E-32	6.39E-02	4.49E-02	9.78E-02	3.42E-02

The best results are written in bold for the purposes of confirming the best algorithm

Table 5 Results of the AFT algorithm and other meta-heuristic algorithms in fixed-dimension multimodal functions

F	AFT		MFO		SOA		CSA		SHO		GWO	
	Ave	Std	Ave	Std	Ave	Std	Ave	Std	Ave	Std	Ave	Std
F_{14}	**9.98E-01**	**0.00E+00**	2.21	1.80	3.35	1.15	**9.98E-01**	8.24E-17	9.68	3.29	3.71	2.15
F_{15}	**3.35E-04**	1.17E-04	1.58E-03	2.48E-04	4.11E-04	4.26E-05	3.38E-04	1.67E-04	9.01E-04	1.06E-04	3.66E-03	3.48E-04
F_{16}	**− 1.0316**	6.77E-16	**− 1.0316**	**0.00E+00**	− 1.08E+01	6.48E-12	**− 1.0316**	6.11E-16	− 1.06E+01	2.86E-11	**− 1.0316**	7.02E-09
F_{17}	**3.97E-01**	**0.00E+00**	3.98E-01	1.13E-03	3.98E-01	1.36E-03	**3.97E-01**	**0.00E+00**	3.98E-01	2.46E-01	3.98E-01	2.42E-02
F_{18}	**2.99**	**1.44E-15**	3.00	1.42E-07	3.00	5.49E-05	**2.99**	1.53E-15	3.00	8.15E-05	3.00	7.16E-06
F_{19}	− 3.86	**2.51E-15**	− 3.86	3.16E-15	− 3.88	3.00E-10	− 3.86	2.65E-15	− 3.75	4.39E-01	− 3.86	1.57E-03
F_{20}	− 2.31	**1.11E-02**	− 3.23	6.65E-02	− 3.32	1.23E-02	− 3.31	5.82E-02	**− 1.44**	5.47E-01	− 3.27	7.27E-02
F_{21}	**− 3.46**	**1.09**	− 1.00E+01	3.52	− 1.00E+01	3.47	− 8.65	3.05	− 1.00E+01	3.80E-01	− 1.01E+01	1.54
F_{22}	**− 6.65**	4.65E-01	− 1.04E+01	3.20	− 1.04E+01	**2.00E-04**	− 10.14E+01	1.39	− 1.04E+01	2.04E-04	− 1.03E+01	2.73E-04
F_{23}	**− 5.77**	1.45E-01	− 1.05E+01	3.68	− 1.05E+01	**1.32E-01**	− 10.31	1.22	− 1.05E+01	2.64E-01	− 1.01E+01	8.17
F	CMA-ES		ACO		PSO		GSA		GA		DE	
	Ave	Std	Ave	Std	Ave	Std	Ave	Std	Ave	Std	Ave	Std
F_{14}	10.23	7.54	**0.998**	3.76E-16	2.77	2.20	3.61	2.96	4.39	4.41E-02	3.97	1.42
F_{15}	0.57E-02	0.0121	7.48E-04	2.18E-04	9.09E-04	2.38E-04	6.84E-03	2.77E-03	7.36E-03	2.39E-04	5.01E-04	**1.00E-04**
F_{16}	**− 1.0316**	6.77E-16	**− 1.0316**	4.82E-16	**− 1.0316**	**0.00E+00**	**− 1.0316**	**0.00E+00**	− 1.04	4.19E-07	**− 1.0316**	3.60E-04
F_{17}	**3.97E-01**	**0.00E+00**	**3.97E-01**	3.09E-04	3.98E-01	4.11E-02	3.98E-01	1.74E-04	3.98E-01	1.00E-04	3.98E-01	8.40E-02
F_{18}	8.40	20.55	3.74	1.62	3.00	6.59E-05	3.01	3.24E-02	3.01	6.33E-07	3.00	9.68E-04
F_{19}	− 3.86	2.70E-15	− 3.860	3.87E-04	− 3.90	3.37E-15	**− 3.22**	4.15E-01	− 3.30	4.37E-10	− 3.40	6.56E-06
F_{20}	− 3.29	0.05	− 3.32	4.80E-16	− 3.32	2.66E-01	− 1.47	5.32E-01	− 2.39	4.37E-01	− 1.79	7.07E-02
F_{21}	− 5.66	3.35	− 8.78	1.47	− 1.00E+01	2.77	− 1.00E+01	1.30	− 1.00E+01	2.34	− 1.00E+01	1.91
F_{22}	− 8.44	3.33	− 9.57	3.80	− 1.04E+01	3.08	− 1.04E+01	2.64	− 1.04E+01	1.37E-02	− 1.04E+01	5.60E-03
F_{23}	− 8.07	3.59	− 9.26	2.88	− 1.05E+01	2.52	− 1.05E+01	2.75	− 1.05E+01	2.37	− 1.05E+01	1.60

The best results are written in bold for the purposes of confirming the best algorithm

best fitness in these two cases (i.e., GSA and SHO). The outcomes in Table 5 indicate that the average fitness values of AFT are better than other algorithms on most of the test functions. With regard to the values of Std, AFT performed better than other algorithms in six out of ten test functions (F_{14}, F_{17} to F_{21}). This supports the fact that we previously concluded that AFT has a high degree of stability when applied to different search spaces. The results in Tables 4 and 5 indicate that AFT is in the first rank, in terms of the average fitness value, in twelve out of sixteen test functions (i.e., F_8, F_{10}, F_{12} to F_{18} and F_{21}–F_{23}). This reinforces that AFT has good exploration ability when it is employed in these search problems.

7.3 Performance of self-adaptive AFT algorithm

The amendment value of the self-adaptive tracking distance of AFT algorithm, or referred to as SAFT, is computed using the ratio of the current tracking distance and perception potential constants of the search agents over the course of iterations. The SAFT algorithm has two main parameters, referred to as θ and ρ, that influence the search performance. These parameters are the adjustment frequency and opportunity of the tracking distance constant, respectively. Small values for θ and ρ may cause the search agent to frequently move, which implies that the search agent will have a difficulty in converging. On the other hand, large values for θ and ρ make the search agent to lose the globally optimum area, and converge to a locally optimum area that leads to early convergence. To locate the best settings for θ and ρ, several experiments were performed on some of the unimodal functions with different values for these parameters, where $\theta \in \{1, 2, 3\}$ and $\rho \in \{0.2, 0.5, 0.8\}$. The results obtained based on these parameter settings after running the experiments 30 times are shown in Table 6.

It is evident from the results shown in Table 6 that the parameter settings $\theta = 3$ and $\rho = 0.2$ are the best settings.

7.4 Performance of hierarchical AFT algorithm

To illustrate the performance of the hierarchical AFT (HAFT) algorithm, it is tested on 23 benchmark functions with various dimensions. The parameter settings of HAFT are given as follows: $T_0 = 1$, $L = 100$, $n = 30$ and $T = 1000$. The experimental results obtained by HAFT which consist of mean and standard deviation values are given in Table 7.

Table 7 presents that HAFT can effectively and efficiently find optimal solutions for many benchmark functions. This indicates that HAFT is efficacious in balancing the search capability of AFT between exploration and exploitation in order to boost its accuracy from the onset to the end of the optimization process. A comparison between HAFT in Table 7 and AFT in Tables 3, 4 and 5 shows that HAFT has better performance.

7.5 Evaluation of the balance of exploitation and exploration of AFT

The multimodal function with fixed dimension F_{16} shown in Eq. 49 is used as an example to illustrate the evaluation of the balance response of AFT.

$$F_{16}(x_1, x_2) = 4x_1^2 - 2.1x_1^4 + \frac{1}{3}x_1^6 + x_1x_2 - 4x_2^2 + 4x_2^4 \tag{49}$$

In Eq. 49, the range of x_1 and x_2 is set to: $-5 \le x_i \le 5$ with the dimension set to 2. Figure 9 shows the performance behavior yielded by AFT in the function F_{16}, over 500 iterations, in terms of evaluating the balance given by Eqs. 24 and 25.

Table 6 Results of SAFT for different values of θ and ρ on some unimodal benchmark functions

	F1	F2	F3	F4	F5
$\theta = 2, \rho = 0.5$	1.35E-71 ± 4.21E-71	1.89E-32 ± 5.13E-32	1.03E-43 ± 3.21E-43	3.11E-27 ± 7.01E-27	1.19E+00 ± 1.92E+00
$\theta = 1, \rho = 0.2$	1.30E-78 ± 3.73E-78	8.33E-34 ± 2.15E-33	1.56E-48 ± 4.62E-48	2.36E-29 ± 3.52E-29	7.97E-01 ± 1.68E+00
$\theta = 1, \rho = 0.5$	3.79E-73 ± 1.00E-72	5.63E-33 ± 1.48E-32	4.01E-42 ± 1.26E-41	1.87E-26 ± 3.85E-26	3.98E-01 ± 1.26E+00
$\theta = 1, \rho = 0.8$	1.28E-65 ± 2.35E-65	1.77E-30 ± 3.23E-30	6.37E-37 ± 1.92E-36	2.26E-24 ± 3.59E-24	3.98E-01 ± 1.26E+00
$\theta = 2, \rho = 0.2$	3.47E-78 ± 9.82E-78	3.38E-34 ± 1.06E-33	2.56E-49 ± 7.18E-49	7.34E-30± 1.50E-29	3.98E-01 ± 1.26E+00
$\theta = 2, \rho = 0.8$	2.20E-65 ± 3.73E-65	1.52E-29 ± 3.21E-29	3.74E-38 ± 7.30E-38	1.14E-23± 2.33E-23	5.18E-15 ± 1.13E-14
$\theta = 3, \rho = 0.2$	5.09E-80 ± 1.09E-79	2.73E-36 ± 4.68E-36	1.84E-49 ± 5.79E-49	5.32E-30 ± 1.05E-29	1.19E+00 ± 1.92E+00
$\theta = 3, \rho = 0.5$	9.49E-73 ± 2.58E-72	3.85E-32 ± 8.39E-32	1.27E-44 ± 2.91E-44	6.89E-27 ± 9.79E-27	7.97E-01 ± 1.68E+00
$\theta = 3, \rho = 0.8$	4.35E-66 ± 1.22E-65	9.73E-30 ± 2.34E-29	3.61E-39 ± 6.10E-39	3.28E-24 ± 7.29E-24	7.97E-01 ± 1.68E+00

Table 7 Experimental results of HAFT algorithm in standard benchmark test functions

F	Ave	Std
F_1	3.19E-74	1.73E-73
F_2	7.65E-33	2.97E-32
F_3	2.32E-45	9.65E-44
F_4	1.63E-25	5.61E-24
F_5	6.19E-01	1.05E-01
F_6	0.00E+00	0.00E+00
F_7	3.12E-01	2.48E-01
F_8	− 1.07E+03	2.17E+02
F_9	1.79E-01	1.41E-01
F_{10}	2.00E-20	1.10E-19
F_{11}	2.00E-01	1.08E-01
F_{12}	3.36E-12	8.37E-12
F_{13}	8.74E-34	4.78E-33
F_{14}	9.98E-01	0.00E+00
F_{15}	3.32E-04	1.13E-04
F_{16}	− 1.0316	6.77E-16
F_{17}	3.97E-01	0.00E+00
F_{18}	2.99	1.00E-15
F_{19}	− 3.86	2.71E-16
F_{20}	− 2.31	1.11E-02
F_{21}	− 3.48	1.19E+00
F_{22}	− 5.01	2.79E-01
F_{23}	− 6.73	1.59E-01

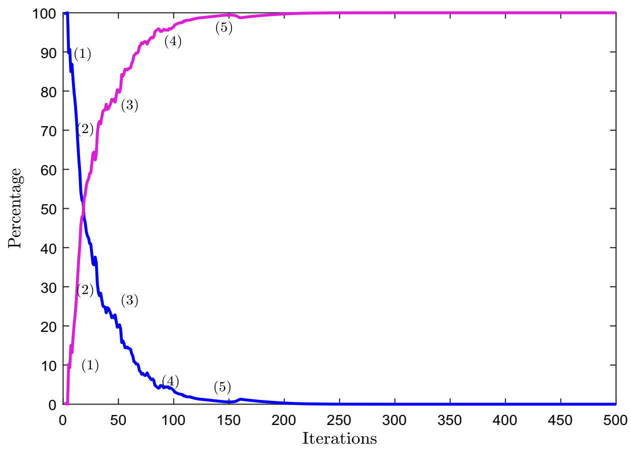

Fig. 9 Performance of AFT during 500 iterations which describes the balance evaluation given by Eqs. 24 and 25

In Fig. 9, five points (1), (2), (3), (4) and (5) have been chosen to represent the diversity of solutions and the balance assessments of each of them. Point (1) represents a premature stage of the AFT algorithm where the balance evaluation values of XPL% and XPT% are 90 and 10, respectively. With these percentage, AFT works with a clear direction to explore the search space. With this effect, it can be inferred that the solutions preserve a high dispersion of the search space. Point (2) correlates with 70 iterations, where at this position, the balance evaluation conserves a value of XPL% = 70 in conjunction with XPT% = 30. In this position, AFT fundamentally conducts exploration with a low degree of exploitation. Points (3) and (4) correspond to 75 and 100 iterations, respectively, where the balance assessments have exploration and exploitation values of XPL%= 25, XPT% = 75, XPL% = 05 and XPT% = 95, respectively. At these percentage, the behavior of AFT was flipped to promote more exploitation than exploration. Under these configurations, the solutions are spread out in several bunches which reduces the overall diversity. Finally, point (5) implements the last juncture of the AFT algorithm. In such a situation, the AFT algorithm sustains a perspicuous trend to the exploitation of the top found solutions without taking into account any exploration strategy.

7.6 Chaotic maps for AFT

This work integrates four chaotic maps into one of the components of AFT. This is to further study and investigate the effectiveness of chaos theory in improving exploration and/or exploitation of AFT. Chaotic maps are applied to define the selection probability of the defined cases of AFT which is directed to promote its performance degree. The four chaotic maps selected in this study are listed in Table 8.

The set of chaotic maps shown in Table 8 was selected with different behaviors with the initial point set to 0.7 for each of them. As mentioned earlier, chaotic maps are used to manipulate the selection process of the defined three cases of the AFT algorithm, where this process was defined by a probability of rand. Here, chaotic maps were used to provide chaotic behaviors for this probability. The final value from the chaotic map should lie within the range of [0, 1]. The proposed Chaotic AFT (CAFT) algorithm is benchmarked on 23 benchmark test functions with chaotic iterative (I), circle (C), logistic (L) and piecewise (P) selection operators. The results are averaged over 30 independent runs. The mean and standard deviation results of the best solutions found at the last iteration are reflected in Table 9.

Considering the results of Table 9, it can be said that chaotic maps improve the performance of AFT not only in terms of exploitation but also exploration. These results are superior to the corresponding ones of the standard AFT as shown in Tables 3, 4 and 5.

Table 8 Four different types of chaotic maps used to improve AFT

No.	Name	Chaotic map	Range
1	Iterative [104]	$x_{i+1} = \sin(\frac{a\pi}{x_i}), a = 0.7$	$(-1, 1)$
2	Circle [105]	$x_{i+1} = \mathrm{mod}\big(x_i + b - (\frac{a}{2\pi})\sin(2\pi x_i), 1\big), a = 0.5, b = 0.2$	
3	Logistic [104]	$x_{i+1} = ax_i(1 - x_i), a = 4$	(0, 1)
4	Piecewise [106]	$x_{i+1} = \begin{cases} \frac{x_i}{P} & 0 \le x_i < P \\ \frac{x_i - P}{0.5 - P} & P \le x_i < 0.5 \\ \frac{1 - P - x_i}{0.5 - P} & 0.5 \le x_i < 1 - P \quad P = 0.4 \\ \frac{1 - x_i}{P} & 1 - P \le x_i < 1 \end{cases}$	(0, 1)

Table 9 Results of the CAFT algorithm on F$_1$ –F$_{23}$ with several chaotic selection operators

F	CAFT-C		CAFT-I		CAFT-L		CAFT-P	
	Ave	Std	Ave	Std	Ave	Std	Ave	Std
F_1	1.08E-67	3.19E-67	2.19E-84	4.20E-84	2.80E-56	7.37E-56	1.21E-46	6.59E-46
F_2	1.03E-29	4.22E-29	3.07E-36	9.46E-36	3.53E-26	1.05E-25	3.07E-21	8.80E-21
F_3	4.93E-31	2.63E-30	1.06E-41	3.11E-41	1.48E-29	4.09E-29	3.01E-14	1.65E-13
F_4	4.94E-20	2.18E-19	5.79E-07	2.52E-06	1.188E-17	2.96E-17	2.71E-04	1.45E-03
F_5	1.12E+00	1.64E+00	4.37E+00	2.27E+00	8.89E-01	1.47E+00	1.036E+01	1.64E+01
F_6	0.00E+00	0.00E+00	0.00E+00	0.00E+00	0.00E+00	0.00E+00	0.00E+00	0.00E+00
F_7	4.84E-01	3.28E-01	5.74E-01	3.15E-01	5.87E-01	2.43E-01	4.99E-01	2.82E-01
F_8	− 3.08E+03	2.46E+02	− 3.07E+03	3.46E+02	− 3.02E+03	2.17E+02	− 3.04E+03	3.34E+02
F_9	1.86E+01	7.21E+00	1.99E+01	7.81E+00	1.86E+01	7.80E+00	2.17E+01	8.20E+00
F_{10}	3.85E-02	2.10E-01	7.70E-02	2.93E-01	6.45E-15	1.79E-15	5.49E-02	3.00E-01
F_{11}	1.79E-01	1.01E-01	1.85E-01	1.10E-01	1.66E-01	9.25E-02	2.00E-01	1.35E-01
F_{12}	1.14E-01	5.69E-01	9.32E-02	3.07E-01	3.71E-30	2.01E-29	9.32E-02	4.01E-01
F_{13}	1.34E-32	5.56E-48	1.34E-32	5.56E-48	1.34E-32	5.56E-48	4.22E-23	2.31E-22
F_{14}	9.98E-01	4.12E-17	9.98E-01	1.00E-16	9.98E-01	5.83E-17	9.98E-01	9.21E-17
F_{15}	4.97E-04	3.70E-04	4.11E-04	3.22E-04	3.68E-04	2.32E-04	4.36E-04	3.59E-04
F_{16}	− 1.0316	6.77E-16	− 1.0316	6.77E-16	− 1.0316	6.71E-16	− 1.0316	6.71E-16
F_{17}	3.978E-01	0.00E+00	3.978E-01	0.00E+00	3.978E-01	0.00E+00	3.978E-01	0.00E+00
F_{18}	2.99E+00	2.03E-15	2.99E+00	4.73E-16	2.99E+00	1.23E-15	2.99E+00	1.30E-15
F_{19}	− 3.862	2.68E-15	− 3.862E+00	2.69E-15	− 3.862E+00	2.71E-15	− 3.862E+00	2.69E-15
F_{20}	− 3.318	2.17E-02	− 3.31E+00	3.01E-02	− 3.314E+00	3.01E-02	− 3.31E+00	2.17E-02
F_{21}	− 5.12	2.53E-01	− 8.29E+00	2.49E+00	− 7.79E+00	2.56E+00	− 7.95E+00	2.55E+00
F_{22}	− 5.99	2.38E-01	− 8.44E+00	2.61E+00	− 8.99E+00	2.37E+00	− 7.43E+00	2.84E+00
F_{23}	− 6.77	1.98E-01	− 9.05E+00	2.51E+00	− 9.45E+00	2.19E+00	− 7.92E+00	2.85E+00

7.7 Analysis of the number of clusters of AFT

To investigate the degree of population interaction of the AFT algorithm using the PIN method reported in [83, 84], it is tested with different numbers of clusters c in the optimization of a group of 23 benchmark functions. The parameter c is set as follows: $c \in 3, 7, 11$. The other parameters of AFT are remained the same during each experiment. The AFT algorithm is run 30 times for each function, and the mean and standard deviation are obtained and displayed in Table 10.

It can be found from Table 10 that the effect of population interaction is reasonably positive and can significantly augment the performance of AFT in optimization.

Table 10 The results obtained by the AFT algorithm with different number of clusters (c)

F	$c = 3$		$c = 7$		$c = 11$	
	Ave	Std	Ave	Std	Ave	Std
F_1	2.64E-78	8.20E-78	5.0E-74	1.0E-73	4.49E-72	8.89E-72
F_2	9.48E-35	1.87E-34	1.136E-32	2.32E-32	6.35E-30	1.62E-29
F_3	1.83E-34	3.11E-34	2.62E-19	8.28E-19	8.81E-12	2.67E-11
F_4	2.26E-06	3.54E-06	2.45E-04	5.23E-04	1.74E-03	3.22E-03
F_5	5.36E+00	2.12E+00	4.87E+01	7.58E+01	4.33E+01	8.35E+01
F_6	0.00E+00	0.00E+00	0.00E+00	0.00E+00	0.00E+00	0.00E+00
F_7	4.72E-01	3.53E-01	4.73E-01	3.24E-01	3.63E-01	2.42E-01
F_8	− 2.90E+03	3.09E+02	− 2.78E+03	2.15E+02	− 2.93E+03	3.39E+02
F_9	2.26E+01	9.05E+00	2.38E+01	6.86E+00	2.18E+01	9.59E+00
F_{10}	9.01E-11	2.84E-10	3.64E-01	6.17E-01	9.81E-01	1.08E+00
F_{11}	1.45E-01	4.65E-02	1.47E-01	3.19E-02	1.67E-01	8.53E-02
F_{12}	5.92E-01	1.76E+00	9.33E-02	1.50E-01	1.07E-01	1.76E-01
F_{13}	1.34E-32	2.88E-48	7.16E-20	2.26E-19	1.09E-03	3.47E-03
F_{14}	9.98E-01	7.40E-17	9.98E-01	0.00E+00	9.98E-01	0.00E+00
F_{15}	3.07E-04	1.08E-19	4.47E-04	4.04E-04	5.51E-04	4.66E-04
F_{16}	− 1.0316	0.00E+00	− 1.0316	1.04E-16	− 1.0316	1.04E-16
F_{17}	3.978E-01	0.00E+00	3.978E-01	0.00E+00	3.978E-01	0.00E+00
F_{18}	2.99	8.24E-16	2.99E+00	7.40E-16	2.99E+00	0.00E+00
F_{19}	− 3.862	9.00E-16	− 3.862E+00	9.36E-16	− 3.862E+00	9.36E-16
F_{20}	− 3.31	3.75E-02	− 3.32E+00	4.68E-16	− 3.32E+00	4.68E-16
F_{21}	− 4.12	2.62E+00	− 8.11E+00	2.62E+00	− 8.11E+00	2.63E+00
F_{22}	− 6.74	2.80E+00	− 7.07E+00	2.89E+00	− 9.20E+00	2.54E+00
F_{23}	− 5.10	3.16E+00	− 9.45E+00	2.28E+00	− 7.83E+00	2.84E+00

The performance of AFT is the best at $c = 3$ according to the mean rank while it is not seriously different from other values of c. According to Tables 10 as well as 3, 4 and 5, it is evident that AFT with the use of PIN is somewhat outperforming the standard AFT in optimization.

7.8 Qualitative analysis of AFT

The qualitative results, including search landscapes, convergence curves, average fitness curves in logarithmic shapes, search history and trajectory of the first individual, associated with the AFT algorithm in solving a selected set of test functions, are shown, for up to 200 iterations, in Fig. 10.

The qualitative metric measures used to plot the curves in Fig. 10 can be interpreted as follows:

- *Convergence curves* these curves are used to describe how well AFT converges toward the optimum global, which demonstrates its ability in fulfilling exploration and exploitation phases. They show the best solutions found after each iteration. It is observed that the fitness value of the convergence curves gets better after each iteration. The convergence curves of the unimodal functions F_1, F_3 and F_5 have some slight differences with those of the multimodal functions F_{11} and F_{13} and the fixed-dimension multimodal function F_{16}. This is mainly ascribed to the fact that unimodal functions have only one global optimum with no local optimum. Broadly, AFT provides a fast convergence response for F_1 and F_3, has a sensible convergence response for F_5 and F_{13}, and shows the optimal convergence for F_{11} and F_{16}. As shown in the convergence curves, the AFT algorithm first explores the search space, and most of the thieves move toward the optimal areas of the search space in the first 50 iterations.
- *Average fitness curves* these curves characterize the average fitness values of the thieves at the end of each iteration. They indicate that, as iterations proceed, the average fitness values decline. This underscores that AFT not only improves the global best fitness of one thief, but also improves the fitness of all thieves.
- *Search history* this drawing shows the search history of the thieves during their search for the global optimum. It can be observed that the sample point of the unimodal functions F_1, F_3 and F_5 are distributed in promising

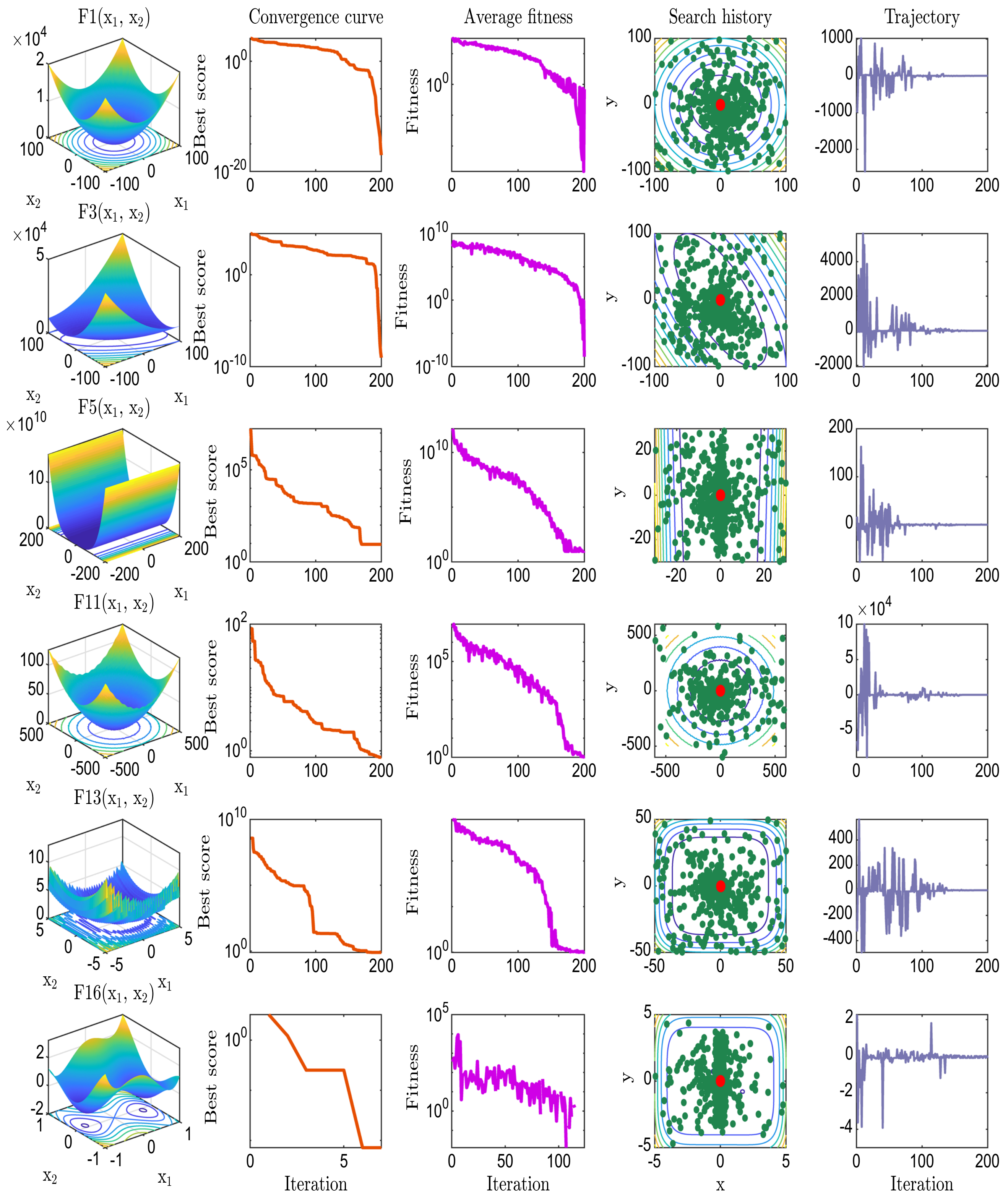

Fig. 10 Qualitative results of AFT for test functions F_1, F_3, F_5, F_{11}, F_{13} and F_{16}: Search landscapes, convergence curves, average fitness curves of all thieves, search histories and trajectories in the first dimension of the first thief

areas. In contrast, some of the sample points of the multimodal and fixed-dimension multimodal functions, F_{11}, F_{13} and F_{16}, respectively, are marginally spread around unpromising areas. This is related to the degree of difficulty of these test functions. As the search history drawings demonstrate, the sample points are roughly scattered around the correct optimum solution, and thus, this constitutes a visual evidence of the ability of AFT in exploitation. This means that the thieves quickly explore the whole search space at first and gradually move toward the global optimum.

- *Trajectory* this plot depicts the thieves' path in the first dimension. Displaying the trajectory of the thieves across all dimensions complicates the analysis process; thus, we have stored and illustrated the search path of the thieves in one dimension. This search plot presents the exploitation and exploration capacities of AFT. Figure 10 shows that the thieves go through abrupt fluctuations in the initial phases, progressive variations after the initial phases, and a steady situation in the final stages. This confirms the ability of AFT in exploring the search space at the initial stages of the search and exploiting the global optimum areas in the consequent stages.

Overall, the metric measures presented in Fig. 10 show that AFT is able to efficiently explore the search space, exploit each promising area, avoid local optimal solutions, achieve high levels of accuracy and reasonably converge toward optimality.

7.9 Performance of AFT in CEC-2017 benchmark

With the aim of further challenging the proposed AFT algorithm, a challenging and a recent benchmark test suite, IEEE CEC-2017, was used. This test suite composes of 30 functions with varying difficulty levels. It should also be pointed out that due to the unstable behavior of the C-2017-f2 function, particularly for higher dimensions, it has been set aside from this test suite [107]. Thus, out of the 29 benchmark test functions, there are 2 unimodal functions (C-2017-f1 and C-2017-f3), 7 simple multimodal functions (C-2017-f4 - C-2017-f10), 10 hybrid functions (C-2017-f11 - C-2017-f20) and 10 composition functions (C-2017-f21 - C-2017-f30). These benchmark test functions include many properties of real-world problems, and thus, in the literature, many algorithms were examined on these test problems. The performance of AFT in this test suite is compared with a set of well-known comparative algorithms previously reported promising performance in the literature. The parameter settings for each optimization algorithm are discussed in Subsect. 7.1 and listed in Table 1.

Table 11 shows the optimization results of AFT and different methods in this test suite. The results of other algorithms were taken from reference [47].

The results in Table 11 bear out the ascendancy of the AFT algorithm over others in optimizing very challenging optimization functions. The AFT algorithm again reported the best average fitness values in twenty-eight out of twenty-nine benchmark test functions. With regard to the *Std* values, AFT performed significantly better than other algorithms in twenty-one out of twenty-nine test functions. This sustains the fact that we formerly inferred that AFT has a high degree of stability when applied to various search domains and test beds. These findings indicate that AFT ranks first in terms of the powerful in exploration and exploitation capabilities, which then turns to obtain high performance in both of the accuracy and standard deviation values. This achievement is a further evidence of the ability of AFT to excel well-studied meta-heuristic algorithms while achieving highly competitive results with high performance methods in complex test functions.

7.10 Performance of AFT in CEC-C06 2019 benchmark

A set of ten benchmark functions, named IEEE CEC-C06 2019, have been developed by Professor Suganthan and his colleagues [108] for use in evaluating single-objective optimization problems. Table 35 in "Appendix C" describes this group of test functions. These benchmark functions are designed to be scalable. The functions CEC04 - CEC10 are shifted and rotated, while CEC01 - CEC03 are neither shifted nor rotated. Table 1 shows the parameter settings of AFT and other optimization algorithms, and a description of these parameter settings is presented in Subsect. 7.1.

The average fitness and standard deviation values were computed to compare between the accuracy of AFT and other algorithms. The results of DA, WOA, and SSA were taken from [109]. Table 12 displays the *Ave* and *Std* values for the AFT algorithm and other algorithms when used to search for the global optimum in the CEC-C06 2019 benchmark test functions.

The outcomes presented in Table 12 show that the AFT algorithm ranked first in six out of ten functions (i.e., CEC01, CEC05 and CEC07-CEC10) in terms of the average fitness. The MFO algorithm is in the second place since it achieved the best *Ave* values in the CEC02 and CEC03 functions, while the MVO algorithm got the best *Ave* in the CEC04 function. Nevertheless, the results obtained by AFT in CEC02, CEC03 and CEC04 are comparable. In terms of measuring the stability of AFT, the value of *Std* was calculated and compared with that of other competitive algorithms. The AFT algorithm gained the least values of *Std* in seven out of ten CEC-C06 functions. This once again confirms that AFT has the most stable results when compared to other algorithms.

Table 11 Optimization results of the AFT algorithm and other algorithms in IEEE CEC-2017 test functions

F	AFT		EO		PSO		GWO		GSA		CMA-ES	
	Ave	Std	Ave	Std	Ave	td	*Ave*	Std	Ave	Std	Ave	Std
C-2017-f1	**100.00**	9.77E-12	2465.30	2206.20	3959.60	4456.60	325132	107351	296.0	275.10	**100.00**	**0.00E+00**
C-2017-f3	**300.00**	2.98E-14	300.00	2.40E-08	300.00	1.90E-10	1538.00	1886.02	10829.20	1620.74	**300.00**	**0.00E+00**
C-2017-f4	**400.00**	4.60E-14	404.48	0.79	405.94	3.28	409.50	7.55	406.60	2.92	**400.00**	**0.00E+00**
C-2017-f5	**509.94**	5.05	510.73	**3.67**	513.06	6.54	513.5	6.10	556.70	8.40	530.18	58.32
C-2017-f6	**600.00**	**2.42E-05**	**600.00**	1.50E-04	600.24	0.98	600.60	0.88	621.60	9.01	682.10	35.43
C-2017-f7	717.39	6.08	720.93	5.74	718.98	5.10	729.80	8.60	714.60	**1.55**	**713.40**	1.63
C-2017-f8	**808.94**	**2.77**	809.51	2.9176	811.39	5.47	814.30	8.26	820.5	4.69	828.90	52.98
C-2017-f9	**900.00**	**3.05E-14**	**900.00**	2.27E-02	**900.00**	5.90E-14	911.30	19.53	**900.00**	6.90E-14	4667.30	2062.80
C-2017-f10	**1260.66**	**206.03**	1418.70	261.63	1473.30	214.97	1530.50	286.67	2694.6	297.62	2588.10	414.47
C-2017-f11	**1101.98**	**1.35E-01**	1105.20	502.18E-02	1110.50	6.28	1140.20	54.13	1134.70	10.45	1111.30	25.44
C-2017-f12	**1265.37**	**715.12**	10340	9790.60	14532	11260	625182	1126443	702723	42075.40	1629.60	198.11
C-2017-f13	**1318.49**	**52.81**	8023.00	6720.8	8601.10	5123.60	9842.30	5633.43	11053	2110.55	1323.62	78.32
C-2017-f14	**1401.98**	**17.98**	1463.30	32.49	1482.10	42.46	3403.53	1953.33	7147.5	1489.52	1452.10	55.98
C-2017-f15	**1504.99**	25.14	1585.60	48.012	1714.30	282.89	3806.60	3860.66	18001	5498.67	1509.63	**16.43**
C-2017-f16	**1600.02**	**15.23**	1649.00	50.915	1860.00	127.65	1725.78	123.85	2149.70	105.80	1815.34	230.13
C-2017-f17	**1711.26**	**9.53**	1731.60	18.071	1761.60	47.50	1759.61	31.29	1857.7	108.32	1830.14	175.83
C-2017-f18	**1825.29**	92.32	12450	11405	14599	11852.20	25806.10	15766.90	8720.50	5060.10	1825.92	**13.53**
C-2017-f19	**1907.48**	**17.38**	1951.50	47.11	2602.80	2185.02	9866.10	6371.09	13670	19168	1920.54	28.68
C-2017-f20	**2008.58**	**21.51**	2020.60	22.28	2085.10	62.25	2075.60	52.04	2272.30	81.72	2494.84	242.65
C-2017-f21	**2200.00**	36.45	2307.50	20.96	2281.70	54.02	2317.10	**7.00**	2357.70	28.20	2324.76	67.76
C-2017-f22	**2200.00**	1.57	2297.40	18.40	2314.80	66.10	2310.10	16.75	2300.00	**7.2E-02**	3532.41	847.62
C-2017-f23	**2300.00**	**8.38**	2615.80	5.5298	2620.80	9.23	2616.40	8.47	2736.50	39.14	2728.8	243.1
C-2017-f24	**2500.00**	105.22	2743.80	6.904	2692.20	108.20	2741.70	8.73	2742.2	**5.52**	2704.43	73.42
C-2017-f25	**2600.02**	19.51	2934.30	19.76	2924.00	25.02	2938.00	23.61	2937.50	**15.36**	2932.01	20.87
C-2017-f26	**2600.00**	97.23	2967.80	164.98	2952.10	249.66	3222.50	427.02	34407.50	628.73	3457.75	598.94
C-2017-f27	**3089.24**	**11.90**	3091.30	2.2414	3116.20	24.99	3104.10	21.81	3259.50	41.66	3137.56	21.37

Table 11 (continued)

F	AFT		EO		PSO		GWO		GSA		CMA-ES	
	Ave	Std	Ave	Std	Ave	td	*Ave*	Std	Ave	Std	Ave	Std
C-2017-f28	**2800.00**	**31.51**	3302.70	133.92	3315.90	121.83	3391.20	101.50	3459.40	33.84	3397.62	131.36
C-2017-f29	**3136.95**	**23.34**	3169.90	24.65	3203.80	52.26	3190.50	42.90	3449.50	171.33	3213.52	109.79
C-2017-f30	3405.88	**4358**	297113	458560	350650	504857	297688	527757	1303361	363843	**304**	444815

The best results are written in bold for the purposes of confirming the best algorithm

Table 12 Results of AFT and other meta-heuristic algorithms in IEEE ECE-C06 2019 benchmark test functions

Functions	AFT		CSA		DA		WOA	
	Ave	Std	Ave	Std	Ave	Std	Ave	Std
CEC01	**3.9876E+04**	**2.3523E+03**	8.9208E+07	1.1887E+08	543E+08	669E+08	411E+08	542E+08
CEC02	17.3428	3.5361E-14	17.3428	1.0562E-05	78.0368	87.7888	17.3495	0.0045
CEC03	12.7024	**3.6134E-15**	12.7024	**3.6134E-15**	13.7026	0.0007	13.7024	0.0
CEC04	45.9001	18.5761	80.7442	41.5589	344.3561	414.0982	394.6754	248.5627
CEC05	**1.1677**	**5.5248E-02**	1.1973	1.1800E01	2.5572	0.3245	2.7342	0.2917
CEC06	7.1860	**1.1781**	9.9243	1.4049	9.8955	1.6404	10.7085	1.0325
CEC07	**110.1464**	**170.3345**	205.0339	1.4647E+02	578.9531	329.3983	490.6843	194.8318
CEC08	**4.1198**	7.8873E-01	4.1259	7.5690E-01	6.8734	0.5015	6.909	**0.4269**
CEC09	**2.3628**	**1.7099E-02**	2.4397	7.5320E-02	6.0467	2.871	5.9371	1.6566
CEC10	**19.9984**	**1.0414E-02**	20.0024	6.0518	21.2604	0.1715	21.2761	0.1111
Functions	MFO		MVO		SCA		SSA	
	Ave	Std	Ave	Std	Ave	Std	Ave	Std
CEC01	1.7580E+10	3.1999E+10	3.0602E+09	1.8495E+09	3.3725E+09	4.2549E+09	605E+08	475E+08
CEC02	**17.3422**	**7.2268e-15**	18.0691	0.35655	17.4672	8.5584e-02	18.3434	0.0005
CEC03	**12.7021**	2.7755E-04	12.7021	2.1233E-09	12.7025	8.3918E-05	13.7025	0.0003
CEC04	169.8126	208.2712	**28.6991**	**11.4953**	1.0590E+03	358.1222	41.6936	22.2191
CEC05	1.2153	0.1522	1.2513	0.10748	2.1658	7.6196E-02	2.2084	0.1064
CEC06	**5.6797**	2.4962	7.5882	1.2560	10.6791	0.6640	6.0798	1.4873
CEC07	409.0922	311.4702	277.6752	171.3962	716.1605	145.8723	410.3964	290.5562
CEC08	5.5186	0.7223	5.1849	0.6046	5.8788	0.4656	6.3723	0.5862
CEC09	2.8346	0.3887	2.3918	2.2188E-02	80.8171	74.9311	3.6704	0.2362
CEC10	20.1601	0.1848	20.0414	6.2121E-02	20.2904	0.7100	21.04	0.078

The best results are written in bold for the purposes of confirming the best algorithm

7.11 Statistical test analysis

The average fitness and standard deviation of the results tabulated in Tables in 3, 4, 5, 11 and 12 presented a general view of the performance of AFT and to what extent AFT was stable during the 30 independent runs. The qualitative analysis presented in Sect. 7.8 clarified the exploitation and exploration skills that the AFT has, but did not interpret to what degree. This subsection presents statistical analysis using Friedman's and Holm's tests [110] to exhibit the statistical significance of the results in Tables 3, 4, 5, 11 and 12 and do not statistically deviate from the results of other competitors.

The significance level (α) in Friedman's test is set to 0.05 in the experiments shown below. In case if the p value calculated by Friedman's test is equal or less than α, a null hypothesis is rejected which indicates that there are significant differences between the performance of the evaluated algorithms. In this study, Friedman's test is followed by Holm's method, as a post hoc test method, to counteract the problem of various comparisons. The lowest ranked method by Friedman's test will be used as a control method for post hoc analysis.

7.11.1 Statistical test on functions F_1–F_7

Table 13 shows a summary of the ranking results generated by Friedman's test when applied to the results presented in Table 3.

The p value given by Friedman's test on the average results of unimodal functions is 4.280198E-05. Thus, the null hypothesis of equivalent accuracy is rejected, confirming the existence of a statistically significant difference between the comparative algorithms. It can be observed from Table 13 that AFT is statistically significant, and it is the best one among all other algorithms. As shown in Table 13, the proposed AFT algorithm achieved the best statistical score on the set of unimodal test functions studied in this work, followed sequentially by SOA, SHO, CMA-ES, GWO, ACO, CSA, PSO, GA, DE, GSA and MFO in the last rank.

After the application of Friedman's test, Holm's test method was used to decide if there were statistically significant differences between the AFT algorithm and the others. Table 14 displays statistical outcomes generated by Holm's method when it is applied to the results shown in Table 13. In Table 14, R_0 represents Friedman's-rank given to the control method (i.e., AFT), R^i represents Friedman's rank given to algorithm i, z represents the statistical difference between two algorithms, and finally ES is the effect size of AFT on algorithm i.

Holm's method in Table 14 rejects the hypotheses that have p-value ≤ 0.00625. It is apparent from Table 14 that AFT yielded the best statistical results. In addition to that, the p-values presented in Table 14 confirm the reliability of the results generated by AFT in unimodal test functions.

Table 13 A summary of the results generated by Friedman's test on the average results of unimodal test functions in Table 3

Algorithm	Rank
AFT	2.714285
MFO	10.0
SOA	2.857142
CSA	6.714285
SHO	3.142857
GWO	4.857142
CMA-ES	4.652321
ACO	5.934213
PSO	6.857142
GSA	7.571428
GA	7.214285
DE	7.285714

7.11.2 Statistical test on functions F_8–F_{23}

Table 15 presents a summary of the ranking results generated by Friedman's test when it is applied to the mean results presented in Table 4.

The p-value generated by Friedman's test on the average fitness values of multimodal functions is 0.091583. There is a statistically significant difference between the comparative algorithms which means that the null hypothesis is rejected. It is clearly observed from the results shown in Table 15 that AFT is statistically significant and that it achieved the best score among other algorithms. In sum, the ranking results from Friedman's test, when applied to multimodal functions in Table 4, are AFT in the first place, followed in order by CMA-ES, GWO, SOA, PSO, GA, DE, SHO, ACO, GSA, CSA and MFO at last.

The statistical results generated by Holm's method on the results presented in Table 15 concerned with to the results of Table 4 are shown in Table16.

In Table 16, the hypotheses with p-value ≤ 0.004545 were rejected by Holm's method. The results in Table 16 confirm that AFT is statistically superior when it is compared to other competitors.

A summary of the ranking results obtained based on applying Friedman's statistical test on the average results in Table 5 is displayed in Table 17.

In Table 17, the proposed AFT outperformed all other algorithms, with the lowest rank of 2.2 followed in order by CSA, CMA-ES, DE, GSA, ACO, SHO, MFO, GWO, PSO and finally SOA.

The statistical results obtained by applying Holm's test method to the results presented in Table 15, which are related to the results reported in Table 5, are shown in Table 18.

In Table 18, the hypotheses with p value ≤ 0.008333 were rejected by Holm's test method. The results in this table illustrate that AFT is statistically the best one, when it is compared to the others.

7.11.3 Statistical test on IEEE CEC-2017 benchmark

Table 19 displays a summary of the statistical results obtained by Friedman's test when it is applied to the average results shown in Table 11.

The p value obtained by Friedman's test to the results of IEEE CEC-2017 test suite is 4.990430E-11. It can be seen from Table 19 that AFT is statistically the best algorithm among all other algorithms. As can be noted from Table 19, AFT has the best statistical score in the IEEE CEC-2017

Table 14 Results of Holm's method according to Friedman's results in Table 13 with $\alpha = 0.05$

i	Method	$z = (R_0 - R_i)/ES$	p value	$\alpha \div i$	Hypothesis
11	MFO	3.780371	1.565945E-4	0.004545	Rejected
10	GSA	2.520247	0.011727	0.005	Rejected
9	DE	2.371997	0.017692	0.005555	Rejected
8	GA	2.334935	0.019546	0.00625	Rejected
7	PSO	2.149623	0.031585	0.007142	Rejected
6	CSA	2.075498	0.037940	0.008333	Rejected
5	ACO	1.916246	0.049940	0.01	Rejected
4	GWO	1.111873	0.266192	0.0125	Not rejected
3	CMA-ES	1.001234	0.665124	0.016666	Rejected
2	SHO	0.222374	0.824022	0.025	Not rejected
1	SOA	0.074124	0.940910	0.05	Not rejected

Table 15 The results generated by Friedman's test on the average results of multimodal functions in Table 4

Algorithm	Rank
AFT	3.966666
MFO	8.666666
SOA	5.25
CSA	7.5
SHO	6.583333
GWO	4.833333
CMA-ES	4.743236
ACO	6.666666
PSO	5.366666
GSA	7.0
GA	5.566666
DE	6.333333

Table 17 A summary of the ranking results obtained based on Friedman's test on the average results shown in Table 5

Algorithm	Rank
AFT	2.2
MFO	7.25
SOA	8.05
CSA	5.1
SHO	7.1
GWO	7.3
CMA-ES	6.33
ACO	7.05
PSO	7.55
GSA	6.75
GA	7.699999
DE	6.649999

Table 16 Results of Holm's method according to Friedman's test results in Table 15 with $\alpha = 0.05$

i	Method	$z = (R_0 - R_i)/\mathrm{SE}$	p value	$\alpha \div i$	Hypothesis
11	MFO	2.161730	0.030638	0.004545	Rejected
10	CSA	1.601281	0.109314	0.005	Not rejected
9	ACO	1.542114	0.138239	0.005555	Rejected
8	GSA	1.361089	0.173485	0.00625	Not rejected
7	SHO	1.1609291	0.245670	0.007142	Not rejected
6	DE	1.040832	0.297953	0.008333	Not rejected
5	SOA	0.520416	0.602773	0.01	Not rejected
4	PSO	0.480384	0.630954	0.0125	Not rejected
3	GWO	0.320256	0.748774	0.016666	Not rejected
2	CMA-ES	0.291537	0.754663	0.025	Not rejected
1	GA	0.500192	0.670181	0.05	Not rejected

test suite, followed in order by CMA-ES, EO, PSO, GWO, and GSA. Holm's test method was then applied to the results in Table 19 to determine if there were statistically significant differences between AFT and other algorithms. The results of this method are presented in Table 20.

It is obvious from Table 20 that the AFT algorithm delivered the best outcomes.

7.11.4 Statistical test on CEC-C06 2019 benchmark

Table 21 displays the ranking results revealed by the application of Friedman's test on the results given in Table 12.

The p-value computed by Friedman's test when it was applied to the average results of CEC-C06 functions in

Table 18 Results of Holm's method according to Friedman's test results in Table 17

i	Method	$z = (R_0 - R_i)/\mathrm{SE}$	p value	$\alpha \div i$	Hypothesis
11	SOA	4.248189	2.155049E-5	0.004545	Rejected
10	GA	4.031128	5.550959E-5	0.005	Rejected
9	PSO	3.938102	8.2128401E-5	0.005555	Rejected
8	GWO	3.783059	1.549124E-4	0.00625	Rejected
7	MFO	3.752050	1.753939E-4	0.007142	Rejected
6	SHO	3.659024	2.531769E-4	0.008333	Rejected
5	ACO	3.598117	3.218995E-4	0.01	Rejected
4	GSA	3.441963	5.775073E-4	0.0125	Rejected
3	DE	3.3799465	7.249993E-4	0.016666	Rejected
2	CMA-ES	2.225894	0.001613	0.025	Rejected
1	CSA	2.418677	0.015577	0.05	Rejected

Table 19 Ranking results of the Friedman's test when applied to the *Ave* results shown in Table 11

Algorithm	Rank
AFT	1.155172
EO	2.534482
PSO	3.482758
GWO	4.448275
GSA	5.051724
CMA-ES	2.2543411

Table 21 The ranking results obtained based on Friedman's test on the IEEE CEC-C06 2019 benchmark with $\alpha = 0.05$

Algorithm	Rank
AFT	1.8
CSA	2.9
DA	6.9
WOA	6.699999
MFO	3.25
MVO	3.049999
SCA	6.0
SSA	5.4

Table 12 is 2.535367E-7. As clearly shown in Table 21, the AFT algorithm is statistically significant as it has the best score among all the other algorithms. The order of the algorithms produced after applying Friedman's test to this test suite is AFT, CSA, MVO, MFO, SSA, SCA, WOA, and DA. The statistical results obtained after applying Holm's method on these benchmark functions are given in Table 22.

As displayed in Table 22, those hypotheses having p-value ≤ 0.016666 were rejected by Holm's method. It can be seen from Table 22 that the AFT algorithm is a robust algorithm in optimizing the IEEE CEC-2017 and IEEE CEC-C06 2019 test suites. It is clearly discerned from the statistical findings presented in this subsection that AFT has better exploitation skills than exploration ones. This can be inferred from the results obtained when the AFT algorithm was applied to unimodal functions compared to the results obtained in multimodal, hybrid and composition functions. However, the results showed that this is not a big concern since the exploration in AFT algorithm is plausible due to the updating mechanism that this algorithm follows to explore the search space to a large extent. A small degree of exploration is usually not sufficient to find the global optimum solution since in optimization problems there is always necessity to strike a convenient balance between exploitation and exploration. These impressive results make the AFT algorithm applicable for solving real-world engineering problems as shown below.

Table 20 Results of Holm's method based on the results reported in Table 19

i	Method	$z = (R_0 - R_i)/\mathrm{SE}$	p value	$\alpha \div i$	Hypothesis
5	GSA	7.931045	2.173091E-15	0.01	Rejected
4	GWO	6.702785	2.044828E-11	0.0125	Rejected
3	PSO	4.737571	2.162948E-6	0.016666	Rejected
2	EO	2.807449	0.004993	0.025	Rejected
1	CMA-ES	2.160424	0.101043	0.05	Not rejected

Table 22 Results of Holm's method based on the Friedman's statistical test results displayed in Table 21

i	Method	$z = (R_0 - R_i)/\mathrm{SE}$	p value	$\alpha \div i$	Hypothesis
7	DA	4.6556417	3.229731E-6	0.007142	Rejected
6	WOA	4.473067	7.710540E-6	0.008333	Rejected
5	SCA	3.834057	1.260464E-4	0.01	Rejected
4	SSA	3.286335	0.001015	0.0125	Rejected
3	MFO	1.323662	0.185615	0.016666	Not rejected
2	MVO	1.141088	0.253833	0.025	Not rejected
1	CSA	1.004158	0.315302	0.05	Not rejected

8 Real engineering design problems

To further substantiate the robustness of the proposed AFT algorithm, its optimization aptitudes were assessed on five real-world engineering design problems, known as: the welded beam problem, the pressure vessel problem, the tension/compression spring problem, the speed reducer problem and the rolling element bearing problem. What distinguishes these design problems is that they have a host of constraints. Hence, in order to be capable of addressing these design problems, it is crucial for the AFT algorithm to be well prepared with a constraint handling method.

8.1 Constraint handling

The AFT algorithm was disposed to fit with a static penalty approach to be talented to deal with the constraints of the aforementioned engineering design problems while addressing them, as elaborated in the following subsections.

$$\zeta(z) = f(z) \pm \left[\sum_{i=1}^{m} l_i \cdot \max(0, t_i(z))^{\alpha} + \sum_{j=1}^{n} o_j \left| U_j(z) \right|^{\beta} \right] \tag{50}$$

where $\zeta(z)$ stands for the objective function, l_i and o_j define positive penalty constants, $t_i(z)$ and $U_j(z)$ are constraint functions. The values of the parameters β and α were set to 2.0 and 1.0, respectively.

The static penalty approach assigns a penalty value for each unattainable solution. This feature encouraged us to use the static penalty function to handle the constrains in the aforementioned design problems, since it can assist the search agents of AFT to move in the direction of the feasible search space for the given problems to be solved. The number of search agents and number of iterations of AFT in solving the following design problems were set to 30 and 1000, respectively.

8.2 Welded beam design problem

The main goal of this design problem is to lessen the manufacturing cost of the welded beam design displayed in Fig. 11 [111].

The elements of the welded beam architecture shown in Figure 11 are a beam, A and the welding desired to be attached to the piece, B. The constraints of this design problem are the bending stress in the beam (θ), buckling load on the bar (P_c), end deflection of the beam (δ) in addition to the shear stress (τ). In addressing this problem to achieve the optimality, it is essential to solve for the structural parameters of the welded beam structure. Specifically, the design variables of this problem can be given as follows: thickness of the weld (h), thickness of the bar (b), length of the clamped bar (l) and the height of the bar (t). The variable vector of this problem can be written as follows: $\mathbf{x} = [x_1, x_2, x_3, x_4]$, where x_1, x_2, x_3 and x_4 represent the values of the variables h, l, t and b, respectively. This cost function required to lessen the cost of designing this problem is formulated in the following form:

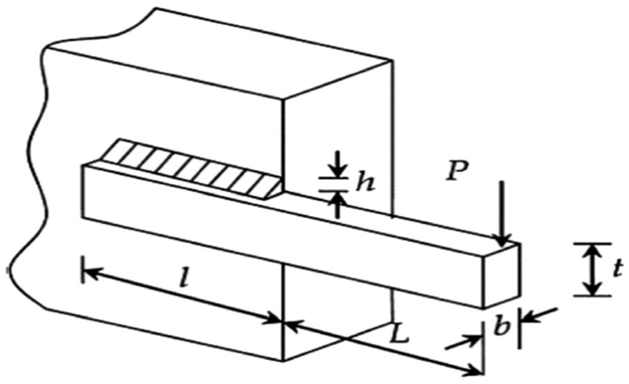

Fig. 11 A schematic diagram of a welded beam design

Minimize:
$f(\mathbf{x}) = 1.10471x_1^2x_2 + 0.04811x_3x_4(14.0 + x_2)$
Subject to the following constraints,
$g_1(\mathbf{x}) = \tau(\mathbf{x}) - \tau_{max} \le 0$
$g_2(\mathbf{x}) = \sigma(\mathbf{x}) - \sigma_{max} \le 0$
$g_3(\mathbf{x}) = x_1 - x_4 \le 0$
$g_4(\mathbf{x}) = 1.10471x_1^2 + 0.04811x_3x_4(14.0 + x_2) - 5.0 \le 0$
$g_5(\mathbf{x}) = 0.125 - x_1 \le 0$
$g_6(\mathbf{x}) = \delta(\mathbf{x}) - \delta_{max} \le 0$
$g_7(\mathbf{x}) = P - P_c(\mathbf{x}) \le 0$
where the remaining variables of the welded beam design are drawn up as follows:
$\tau(\mathbf{x}) = \sqrt{((\tau')^2 + (\tau'')^2) + \frac{2\tau'\tau'' x_2}{2R}}, \tau' = \frac{p}{\sqrt{2}x_1x_2}$
$\tau'' = \frac{MR}{J}, M = P(L + \frac{x_2}{2}), R = \sqrt{(\frac{x_1+x_3}{2})^2 + \frac{x_2^2}{4}}$

$$J = 2\left\{\sqrt{2}x_1x_2\left[\frac{x_2^2}{12} + (\frac{x_1 + x_3}{2})^2\right]\right\}, \sigma(\mathbf{x}) = \frac{6PL}{x_4x_3^2}$$

$$\delta(\mathbf{x}) = \frac{4PL^3}{Ex_4x_3^3}, P_c(\mathbf{x}) = \frac{4.013\sqrt{EGx_3^2x_4^6/36}}{L^2}\left(1 - \frac{x^3}{2L}\sqrt{\frac{E}{4G}}\right)$$

where $P = 6000lb, L = 14$in, $\delta_{max} = 0.25$inch, $E = 30 * 10^6$ psi, $G = 12 * 10^6$ psi, $\delta_{\max} = 13600psi$, $\sigma_{\max} = 30000$ psi. The ranges of variables of this design are used as: $0.1 \le x_i \le 2.0$ when $i = 1$ and 4 and $0.1 \le x_i \le 10.0$ when $i = 2$ and 3.

A comparison of the best solutions obtained from AFT and other optimization algorithms reported in the literature is presented in Table 23.

Succinctly, the results of Table 23 indicate that the AFT algorithm converges toward the optimal design and offers the best solution among all other competitors. A statistical comparison of AFT with other competitors, after 30 independent runs, in terms of the best score, worst score, average score *Ave* and standard deviation score *Std*, is presented in Table 24.

The results of Table 24 expose that AFT again behaves much better than other optimization algorithms in terms of average and *Std* results. This ascertains the degree of reliability of the AFT algorithm in addressing this design problem.

8.3 Pressure vessel design problem

The pressure vessel design is another well-respected engineering problem that has been extensively considered in optimization [101]. The goal of designing this problem is to reduce the overall cost of formation represented as the material and welding of a cylindrical vessel. This is wrapped at both endings with hemispherical heads. Figure 12 illustrates a description of a schematic diagram of this design.

The variables of this design problem can be described as given: The first variable is the inner radius (R), thickness of the shell (T_s), length of the cylindrical section of the vessel without considering the head (L) and the last variable is the thickness of the head (T_h). These variables can be outlined by a vector as follows: $\mathbf{x} = [x_1, x_2, x_3, x_4]$, where the parameters of this vector stand for the values of T_s, T_h, R and L, respectively. The problem of the pressure vessel design can be mathematically defined as follows:

$$\text{Minimize}: f(\mathbf{x}) = 0.6224x_1x_3x_4 + 1.7781x_2x_3^2 + 3.1661x_1^2x_4 + 19.84x_1^2x_3$$

This problem undergoes to four constraints, as outlined below,
$g_1(\mathbf{x}) = -x_1 + 0.0193x_3 \le 0$
$g_2(\mathbf{x}) = -x_2 + 0.00954x_3 \le 0$
$g_3(\mathbf{x}) = -\pi x_3^2x_4 - \frac{4}{3}\pi x_3^3 + 1296000 \le 0$

Table 23 A comparison of the optimal costs attained by AFT and other optimization algorithms for the welded beam problem

Algorithm	Optimal values for variables				Optimum cost
	h	*l*	*t*	*b*	
AFT	0.205729	3.470488	9.036623	0.205729	**1.724852**
SHO [99]	0.205563	3.474846	9.035799	0.205811	1.725661
GWO [100]	0.205678	3.475403	9.036964	0.206229	1.726995
PSO [85]	0.197411	3.315061	10.00000	0.201395	1.820395
MVO [45]	0.205611	3.472103	9.040931	0.205709	1.725472
SCA [101]	0.204695	3.536291	9.004290	0.210025	1.759173
GSA [44]	0.147098	5.490744	10.00000	0.217725	2.172858
GA [102]	0.164171	4.032541	10.00000	0.223647	1.873971
DE [87]	0.206487	3.635872	10.00000	0.203249	1.836250

The best result is written in bold for the purposes of confirming the best algorithm

Table 24 Statistical results obtained from AFT and other algorithms for the welded beam design problem

Algorithm	Best	Ave	Worst	*Std*
AFT	1.724852	1.724852	.724852	1.054459E-15
SHO [99]	1.725661	1.725828	1.726064	0.000287
GWO [100]	1.726995	1.727128	1.727564	0.001157
PSO [85]	1.820395	2.230310	3.048231	0.324525
MVO [45]	1.725472	1.729680	1.741651	0.004866
SCA [101]	1.759173	1.817657	1.873408	0.027543
GSA [44]	2.172858	2.544239	3.003657	0.255859
GA [102]	1.873971	2.119240	2.320125	0.034820
DE [87]	1.836250	1.363527	2.035247	0.139485

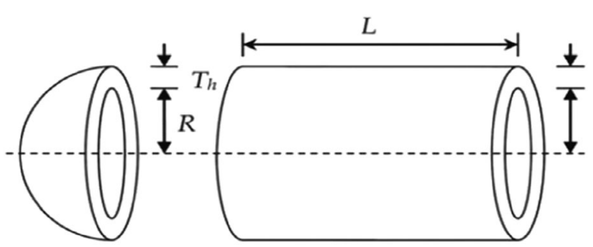

Fig. 12 A structural representation of the cross section of a pressure vessel design

$g4(\mathbf{x}) = x_4 - 240 \le 0$

where $0 \le x1 \le 99$, $0 \le x_2 \le 99$, $10 \le x_3 \le 200$ and $10 \le x_4 \le 200$.

A comparison of the optimal solutions obtained from the proposed AFT algorithm and other optimization algorithms reported in the literature is given in Table 25.

As distinctly perused from Table 25, the AFT algorithm provided the best design with the minimum cost of about 5885.332773. No other competitor algorithm was capable to achieve this cost.

The statistical results of the AFT algorithm for the pressure vessel design problem compared to others, in terms of the best, average, worst and standard deviation scores after 30 independent runs, are presented in Table 26.

The statistical results in Table 26 assert that the AFT algorithm performs better than all other competitors' algorithms in terms of the best, *Ave* and *Std* values obtained so far.

8.4 Tension–compression spring design problem

The structure of a tension/compression spring design problem is shown in Fig. 13 [112].

In the tension/compression spring design problem, we strive to reduce the weight of this design. This optimization problem is subject to three constraints, which are given as follows: surge frequency, shear stress and minimum deflection. The variables of this design problem are the mean coil diameter (D), wire diameter (d) and the number of active coils (N). These variables can be characterized as: $\mathbf{x} = [x_1, x_2, x_3]$, with the parameters of $\mathbf{x}$ represent D, d and N, respectively. The mathematical representation of this problem can be described in the following way:

Minimize: $f(\mathbf{x}) = (x_3 + 2)x_2x_1^2$

This problem is subject to the following constraints:

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

$$g_2(\mathbf{x}) = \frac{4x_2^2 - x_1x_2}{12566(x_2x_1^3 - x_1^4)} + \frac{1}{5108x_1^2} - 1 \le 0$$

$g_3(\mathbf{x}) = 1 - \frac{140.45x_1}{x_2^2 x_3} \le 0$

$g_4(\mathbf{x}) = \frac{x_1 + x_2}{1.5} - 1 \le 0$

where $0.05 \le x_1 \le 2.0$, $0.25 \le x_2 \le 1.3$ and $2 \le x_3 \le 15.0$.

Table 27 shows a comparison between the AFT algorithm presented in this work and other algorithms reported in the literature.

Table 25 A comparison of the results achieved by AFT and other algorithms for the pressure vessel design problem

Algorithm	Optimal values for variables				Optimum cost
	T_s	T_h	R	L	
AFT	12.450698	6.154386	40.319618	199.999999	**5885.332773**
SHO [99]	0.778210	0.384889	40.315040	200.00000	5885.5773
GWO [100]	0.779035	0.384660	40.327793	199.65029	5889.3689
PSO [85]	0.778961	0.384683	40.320913	200.00000	5891.3879
MVO [45]	0.845719	0.418564	43.816270	156.38164	6011.5148
SCA [101]	0.817577	0.417932	41.74939	183.57270	6137.3724
GSA [44]	1.085800	0.949614	49.345231	169.48741	11550.2976
GA [102]	0.752362	0.399540	40.452514	198.00268	5890.3279
DE [87]	1.099523	0.906579	44.456397	179.65887	6550.0230

The best result is written in bold for the purposes of confirming the best algorithm

Table 26 Statistical results obtained from the AFT algorithm and other algorithms in solving the pressure vessel design problem

Algorithm	Best	Ave	Worst	*Std*
AFT	5885.332773	5885.332773	5885.332773	4.178081E-12
SHO [99]	5885.5773	5887.4441	5892.3207	2.893
GWO [100]	5889.3689	5891.5247	5894.6238	013.910
PSO [85]	5891.3879	6531.5032	7394.5879	534.119
MVO [45]	6011.5148	6477.3050	7250.9170	327.007
SCA [101]	6137.3724	6326.7606	6512.3541	126.609
GSA [44]	11550.2976	23342.2909	33226.2526	5790.625
GA [102]	5890.3279	6264.0053	7005.7500	496.128
DE [87]	6550.0230	6643.9870	8005.4397	657.523

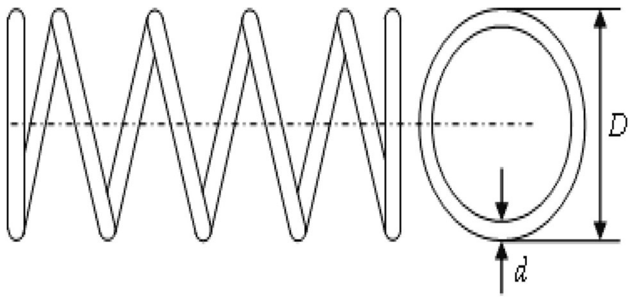

Fig. 13 A schematic diagram of a tension/compression spring design

Table 27 A comparison of the best solutions achieved by AFT and other algorithms for the tension/compression spring design problem

Algorithm	Optimum variables			Optimum weight
	d	*D*	*N*	
AFT	0.051691	0.356777	11.285441	**0.012665**
SHO [99]	0.051144	0.343751	12.0955	0.012674000
GWO [100]	0.050178	0.341541	12.07349	0.012678321
PSO [85]	0.05000	0.310414	15.0000	0.013192580
MVO [45]	0.05000	0.315956	14.22623	0.012816930
SCA [101]	0.050780	0.334779	12.72269	0.012709667
GSA [44]	0.05000	0.317312	14.22867	0.012873881
GA [102]	0.05010	0.310111	14.0000	0.013036251
DE [87]	0.05025	0.316351	15.23960	0.012776352

The best result is written in bold for the purposes of confirming the best algorithm

Examining the results of Table 27 in terms of the optimal costs, we can clearly realize that AFT scored the best solution out of all the optimization algorithms for this problem and achieved the best design with a cost of 0.012665, which no other competitor has achieved. A comparison of the statistical results obtained by AFT and other algorithms reported in the literature for this design problem is presented in Table 28.

The results in Table 28 divulge that the AFT algorithm once again acts much better in terms of the statistical results than other algorithms.

8.5 Speed reducer design problem

The speed reducer design problem is a challenging benchmark problem due to that this problem comprises of seven variables [113]. The structural description of a speed reducer design is shown in Fig. 14.

The prime objective of this problem is to reduce the weight of the speed reducer, which is subject to the following constraints: transverse deflections of the shafts, surface stress, bending stress of the gear teeth and stresses in the shafts [99]. The seven variables of this problem can be given as follows: the number of teeth in the pinion (z), the module of teeth (m), the face width (b), the module of the teeth (m), the diameter of the first shafts (d_1), the diameter of the second shafts (d_2), the length of the first shaft between bearings (l_1) and the last design variable is the length of the second shaft between bearings (l_2). These variables can be represented as follows: $\mathbf{x} = [x_1 x_2 x_3 x_4 x_5 x_6 x_7]$. The mathematical description of this design problem can be given as follows:

Minimize:

$$f(\mathbf{x}) = 0.7854x_1x_2^2(3.3333x_3^2 + 14.9334x_3 - 43.0934) - 1.508x_1(x_6^2 + x_7^2) + 7.4777(x_6^3 + x_7^3) + 0.7854(x_4x_6^2 + x_5x_7^2)$$

This function is subject to eleven constraints, which are described as follows:

$$g_1(\mathbf{x}) = \frac{27}{x_1x_2^2x_3} - 1 \leq 0$$

$$g_2(\mathbf{x}) = \frac{397.5}{x_1x_2^2x_3^2} - 1 \leq 0$$

$$g_3(\mathbf{x}) = \frac{1.9x_4^3}{x_2x_6^4x_3} - 1 \leq 0$$

$$g_4(\mathbf{x}) = \frac{1.93x_5^3}{x_2x_7^4x_3} - 1 \leq 0$$

$$g_5(\mathbf{x}) = \frac{\left[(745(x_4/x_2x_3))^2 + 16.9 \times 10^6\right]^{1/2}}{110x_6^3} - 1 \leq 0$$

Table 28 Statistical results of the proposed AFT algorithm and other optimization algorithms for the tension/compression spring design problem

Algorithm	Best	Ave	Worst	*Std*
AFT	0.012665	0.012665	0.012665	3.216681E-10
SHO [99]	0.012674000	0.012684106	0.012715185	0.000027
GWO [100]	0.012678321	0.012697116	0.012720757	0.000041
PSO [85]	0.013192580	0.014817181	0.017862507	0.002272
MVO [45]	0.012816930	0.014464372	0.017839737	0.001622
SCA [101]	0.012709667	0.012839637	0.012998448	0.000078
GSA [44]	0.012873881	0.013438871	0.014211731	0.000287
GA [102]	0.013036251	0.014036254	0.016251423	0.002073
DE [87]	0.012776352	0.013069872	0.015214230	0.000375

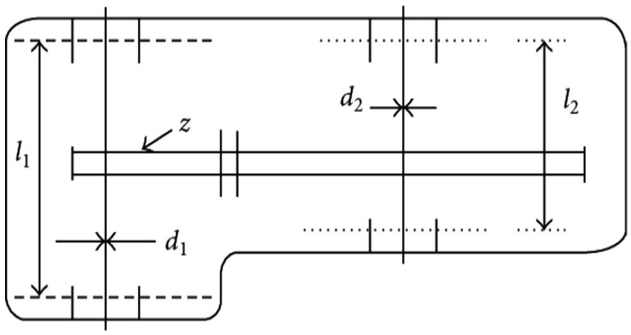

Fig. 14 A structural design of a speed reducer problem

$$g_6(\mathbf{x}) = \frac{\left[(745(x_5/x_2x_3))^2 + 157.5 \times 10^6\right]^{1/2}}{85x_7^3} - 1 \le 0$$

$g_7(\mathbf{x}) = \frac{x_2x_3}{40} - 1 \le 0$
$g_8(\mathbf{x}) = \frac{5x_2}{x_1} - 1 \le 0$
$g_9(\mathbf{x}) = \frac{x_1}{12x_2} - 1 \le 0$
$g_{10}(\mathbf{x}) = \frac{1.5x_6+1.9}{x_4} - 1 \le 0$
$g_{11}\mathbf{x}) = \frac{1.1x_7+1.9}{x_5} - 1 \le 0$

where the range of the design variables b, m, z, l_1, l_2, d_1 and d_2 were used as $2.6 \le x_1 \le 3.6$, $0.7 \le x_2 \le 0.8$, $17 \le x_3 \le 28$, $7.3 \le x_4 \le 8.3$, $7.3 \le x_5 \le 8.3$, $2.9 \le x_6 \le 3.9$ and $5.0 \le x_4 \le 5.5$, respectively.

Table 29 displays the best designs and optimum costs achieved by the AFT algorithm and other algorithms for the speed reducer design problem.

Table 29 corroborates that AFT provides the optimal design compared to others with a cost of approximately 2994.471066. The statistical results of AFT and other competitive methods, over 30 independent runs, for the speed reducer design problem are tabulated in Table 30.

According to the statistical results in Table 30, the AFT algorithm found the best results compared to other promising optimization algorithms.

8.6 Rolling Element Bearing Design Problem

The main goal of this design problem is to make the dynamic load carrying power of a rolling element bearing as large as possible. The schematic diagram of this design problem is shown in Fig. 15 [99].

This problem consists of ten decision variables given as follows: ball diameter (D_b), pitch diameter (D_m), number of balls (X), inner (f_i) and outer (f_o) raceway curvature factors, K_{Dmin}, K_{Dmax}, e, ϵ and ζ. The mathematical representation of this design problem is as follows:

Maximize: $C_d = f_c X^{2/3} D_b^{1.8} \quad if \;\; D \le 25.4\text{mm}$

$$C_d = 3.647 f_c X^{2/3} D_b^{1.4} \quad if \;\; D > 25.4\,\text{mm}$$

The constraints and f_c of this design problem are presented as follows:

$g_1(\mathbf{x}) = \frac{\phi_0}{2sin^{-1}(D_b/D_m)} - X + 1 \le 0$
$g_2(\mathbf{x}) = 2D_b - K_{Dmin}(D - d) \ge 0$
$g_3(\mathbf{x}) = K_{Dmax}(D - d) - 2D_b \ge 0$
$g_4(\mathbf{x}) = \zeta B_w - D_b \le 0$
$g_5(\mathbf{x}) = D_m - 0.5(D + d) \ge 0$
$g_6(\mathbf{x}) = (0.5 + e)(D + d) - D_m \ge 0$
$g_7(\mathbf{x}) = 0.5(D - D_m - D_b) - \epsilon D_b \ge 0$
$g_8(\mathbf{x}) = f_i \ge 0.515$
$g_9(\mathbf{x}) = f_o \ge 0.515$

$$f_c = 37.91\left[1 + \left\{1.04\left(\frac{1-\gamma}{1+\gamma}\right)^{1.72}\left(\frac{f_i(2f_o-1)}{f_o(2f_i-1)}\right)^{0.41}\right\}^{10/3}\right]^{-0.3} \times \left[\frac{\gamma^{0.3}(1-\gamma)^{1.39}}{(1+\gamma)^{1/3}}\right]\left[\frac{2f_i}{2f_i-1}\right]^{0.41}$$
$$x = \left\{\frac{(D-d)}{2} - 3\frac{T}{4})\right\}^2 + \left\{\frac{D}{2} - \frac{T}{4} - D_b\right\}^2 - \left\{\frac{d}{2} + \frac{T}{4}\right\}^2$$
$$y = 2\left\{\frac{(D-d)}{2} - 3\frac{T}{4}\right\}\left\{\frac{D}{2} - \frac{T}{4} - D_b\right\} \tag{51}$$

Table 29 A comparison of the best results obtained by AFT and other algorithms for the speed reducer design problem

Algorithm	Optimum variables							Optimum cost
	b	m	z	l_1	l_2	d_1	d_2	
AFT	3.5	0.69	17.	7.3	7.715319	3.350214	5.286654	**2994.471066**
SHO [99]	3.50159	0.7	17	7.3	7.8	3.35127	5.28874	2998.5507
GWO [100]	3.506690	0.7	17	7.380933	7.815726	3.357847	5.286768	3001.288
PSO [85]	3.500019	0.7	17	8.3	7.8	3.352412	5.286715	3005.763
MVO [45]	3.508502	0.7	17	7.392843	7.816034	3.358073	5.286777	3002.928
SCA [101]	3.508755	0.7	17	7.3	7.8	3.461020	5.289213	3030.563
GSA [44]	3.600000	0.7	17	8.3	7.8	3.369658	5.289224	3051.120
GA [102]	3.510253	0.7	17	8.35	7.8	3.362201	5.287723	3067.561
DE [87]	3.520124	0.7	17	8.37	7.8	3.366970	5.288719	3029.002

The best result is written in bold for the purposes of confirming the best algorithm

Table 30 Statistical results of the AFT algorithm and other optimization algorithms for the speed reducer design problem

Algorithm	Best	Ave	Worst	*Std*
AFT	2994.471066	2994.471066	2994.471073	1.419722E-06
SHO [99]	2998.5507	2999.640	3003.889	1.93193
GWO [100]	3001.288	3005.845	3008.752	5.83794
PSO [85]	3005.763	3105.252	3211.174	79.6381
MVO [45]	3002.928	3028.841	3060.958	13.0186
SCA [101]	3030.563	3065.917	3104.779	18.0742
GSA [44]	3051.120	3170.334	3363.873	92.5726
GA [102]	3067.561	3186.523	3313.199	17.1186
DE [87]	3029.002	3295.329	3619.465	57.0235

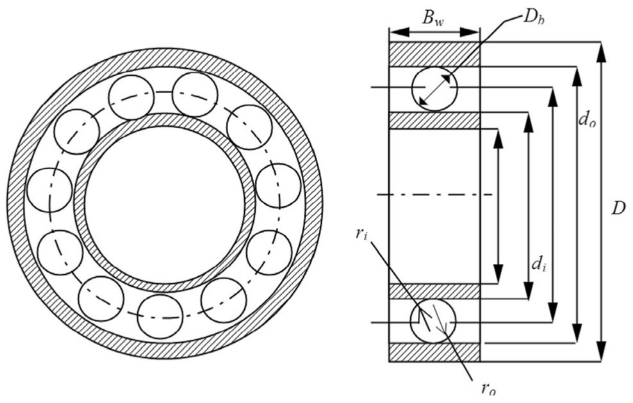

Fig. 15 A schematic view of a rolling element bearing

$$\phi_0 = 2\pi - 2cos^{-1}\left(\frac{x}{y}\right)$$
$$\gamma = \frac{D_b}{D_m}, f_i = \frac{r_i}{D_b}, f_o = \frac{r_o}{D_b}, T = D - d - 2D_b$$
$$D = 160, d = 90, B_w = 30, r_i = r_o = 11.033$$
$$0.5(D+d) \le D_m \le 0.6(D+d),$$
$$0.15(D-d) \le D_b \le 0.45(D-d)$$
$$4 \le X \le 50, 0.515 \le f_i \text{ and } f_o \le 0.6$$
$$0.4 \le K_{Dmin} \le 0.5, 0.6 \le K_{Dmax} \le 0.7$$
$$0.3 \le e \le 0.4, 0.02 \le e \le 0.1, 0.6 \le \zeta \le 0.85$$

Table 31 shows a comparison of the best solutions for the rolling element bearing design obtained by AFT and other optimization algorithms.

As per the optimum costs of the rolling element bearing design problem reported in Table 31, the AFT algorithm got the best design with the optimal cost of about 85206.641. The statistical results of AFT and other optimization methods over 30 runs are shown in Table 32.

It may be observed from Table 32 that the AFT algorithm has once again obtained the best optimal solutions for the rolling element bearing design problem over other algorithms.

In a nutshell, the general performance of the proposed AFT algorithm has corroborated its reliability and efficiency in addressing the above five classical engineering design problems. Therefore, we can deduce that the AFT algorithm is an appropriate and effective optimizer and is definitely a promising candidate for solving real-world contemporary problems.

9 Conclusion and Future Work

This paper has proposed a novel human-based meta-heuristic algorithm called Ali Baba and the forty thieves (AFT) for solving global optimization problems. The performance of the AFT algorithm was benchmarked on three benchmarks of sixty-two basic and challenging test functions taken from the so-called classic benchmark functions,

Table 31 Optimization results of the rolling element bearing design problem achieved by AFT and other algorithms

Algorithm	Optimum variables										Optimum cost
	D_m	D_b	X	f_i	f_o	K_{Dmin}	K_{Dmax}	ϵ	e	ζ	
AFT	125	21.418	11.356	0.515	0.515	0.4	0.680	0.3	0.02	0.622	**85206.641**
SHO [99]	125	21.407	10.932	0.515	0.515	0.4	0.7	0.3	0.02	0.6	85054.532
GWO [100]	125	21.351	10.987	0.515	0.515	0.5	0.688	0.300	0.032	0.627	84807.111
PSO [85]	125	20.753	11.173	0.515	0.515	0.5	0.615	0.3	0.051	0.6	81691.202
MVO [45]	125	21.322	10.973	0.515	0.515	0.5	0.687	0.301	0.036	0.610	84491.266
SCA [101]	125	21.148	10.969	0.515	0.515	0.5	0.7	0.3	0.027	0.629	83431.117
GSA [44]	125	20.854	11.149	0.515	0.517	0.5	0.618	0.304	0.02	0.624	82276.941
GA [102]	125	20.775	11.012	0.515	0.515	0.5	0.613	0.3	0.050	0.610	82773.982
DE [87]	125	20.871	11.166	0.515	0.516	0.5	0.619	0.301	0.050	0.614	81569.527

The best result is written in bold for the purposes of confirming the best algorithm

Table 32 Statistical results obtained from the AFT algorithm and others for the rolling element bearing design problem

Algorithm	Best	Ave	Worst	*Std*
AFT	85206.201105	85206.641021	85207.012231	1.290128E-03
SHO [99]	85054.532	85024.858	85853.876	0186.68
GWO [100]	84807.111	84791.613	84517.923	0137.186
PSO [85]	81691.202	50435.017	32761.546	13962.150
MVO [45]	84491.266	84353.685	84100.834	0392.431
SCA [101]	83431.117	81005.232	77992.482	1710.777
GSA [44]	82276.941	78002.107	71043.110	3119.904
GA [102]	82773.982	81198.753	80687.239	1679.367
DE [87]	81569.527	80397.998	79412.779	1756.902

IEEE CEC-2017 benchmark functions and IEEE CEC-C06 2019 benchmark functions. Several developments were conducted for the AFT algorithm from several aspects to innervate its exploration and exploitation aptitudes. Extensive comparisons with many well-studied, new and high-performance algorithms have shown that AFT is highly reliable and effective in getting near-optimal or optimal solutions for most of the test functions studied. In real-world problems, the AFT algorithm was practically applied to solve five engineering design problems as evidence of its reliability and applicability in addressing real-life applications. In future work, a parallel optimization algorithm could be developed by a combination of AFT algorithm and other algorithms as potential researches to further improve its performance level. Further expansions of the AFT algorithm can be developed by adapting, implementing and testing both binary and multi-objective version of this algorithm to solve large-scale real-world problems.

Appendix A. Unimodal, multimodal and fixed-dimension multimodal functions

A detailed description of the unimodal benchmark functions (F_1–F_7), multimodal benchmark functions (F_8–F_{13}) and fixed-dimension multimodal benchmark functions (F_{14}–F_{23}) is tabulated in Table 33.

Table 33 Characteristics of the unimodal, multimodal and fixed-dimension multimodal functions used in this work

F	Function formulation	$f_{\min}$	Class	Dim	Range
F_1	$\sum_{i=1}^{n} x_i^2$	0	U	10	$x_i \in [-100{,}100]$
F_2	$\sum_{i=1}^{n} \lvert x_i\rvert + \prod_{i=1}^{n} \lvert x_i\rvert$	0	U	10	$x_i \in [-10{,}10]$
F_3	$\sum_{i=1}^{n}\left(\sum_{j-1}^{i} x_j\right)^2$	0	U	10	$x_i \in [-100{,}100]$
F_4	$\max_i\{\lvert x_i\rvert, 1 \le i \le n\}$	0	U	10	$x_i \in [-100{,}100]$
F_5	$\sum_{i=1}^{n-1}\left[100\left(x_{i+1}-x_i^2\right)^2+(x_i-1)^2\right]$	0	U	10	$x_i \in [-30{,}30]$
F_6	$\sum_{i=1}^{n}([x_i+0.5])^2$	0	U	10	$x_i \in [-100{,}100]$
F_7	$\sum_{i=1}^{n} i x_i^4 + \mathrm{random}[0,1)$	0	U	10	$x_i \in [-1.28{,}1.28]$
F_8	$\sum_{i=1}^{n} -x_i \sin\left(\sqrt{\lvert x_i\rvert}\right)$	-418E+5	M	10	$x_i \in [-500{,}500]$
F_9	$\sum_{i=1}^{n}\left[x_i^2 - 10\cos(2\pi x_i)+10\right]$	0	M	10	$x_i \in [-5.12{,}5.12]$
F_{10}	$-20\exp\left(-0.2\sqrt{\frac{1}{n}\sum_{i=1}^{n} x_i^2}\right) - \exp\left(\frac{1}{n}\sum_{i=1}^{n} cos(2\pi x_i)\right) + 20 + e$	0	M	10	$x_i \in [-32{,}32]$
F_{11}	$\frac{1}{4000}\sum_{i=1}^{n} x_i^2 - \prod_{i=1}^{n}\cos\left(\frac{x_i}{\sqrt{i}}\right)+1$	0	M	10	$x_i \in [-600{,}600]$
F_{12}	$\frac{\pi}{n}\left\{10\sin(\pi y_1) + \sum_{i=1}^{n-1}(y_i-1)^2\left[1+10\sin^2(\pi y_{i+1})\right] + (y_n-1)^2\right\} + \sum_{i=1}^{n} u(x_i,10,100,4)$	0	M	10	$x_i \in [-50{,}50]$
	$y_i = 1+\frac{x_i+1}{4} u(x_i,a,k,m) = \begin{cases} k(x_i-a)^m & x_i > a \\ 0-a & < x_i < a \\ k(-x_i-a)^m & x_i < -a \end{cases}$				
F_{13}	$0.1\left\{\sin^2(3\pi x_1) + \sum_{i=1}^{n}(x_i-1)^2\left[1+\sin^2(3\pi x_i+1)\right] + (x_n-1)^2\left[1+\sin^2(2\pi x_n)\right]\right\} + \sum_{i=1}^{n} u(x_i,5,100,4)$	0	M	10	$x_i \in [-50{,}50]$
F_{14}	$\left(\frac{1}{500} + \sum_{j=1}^{25}\frac{1}{j+\sum_{i=1}^{2}\left(x_i-a_{ij}\right)^6}\right)^{-1}$	1	F	2	$x_i \in [-65.536, 65.536]$
F_{15}	$\sum_{i=1}^{11}\left[a_i - \frac{x_1\left(b_i^2+b_i x_2\right)}{b_i^2+b_i x_3+x_4}\right]^2$	0.0003	F	4	$x_i \in [-5, 5]$
F_{16}	$4x_1^2 - 2.1x_1^4 + \frac{1}{3}x_1^6 + x_1 x_2 - 4x_2^2 + 4x_2^4$	−1.0316	F	2	$x_i \in [-5, 5]$
F_{17}	$\left(x_2 - \frac{5.1}{4\pi^2}x_1^2 + \frac{5}{\pi}x_1 - 6\right)^2 + 10\left(1-\frac{1}{8\pi}\right)\cos x_1 + 10$	0.398	F	2	$x_i \in [-5, 0]$ & $[10, 15]$
F_{18}	$\left[1 + (x_1+x_2+1)^2\left(19 - 14x_1 + 3x_1^2 - 14x_2 + 6x_1x_2 + 3x_2^2\right)\right]$	3	F	2	$x_i \in [-2, 2]$
	$\times\left[30 + (2x_1-3x_2)^2 \times \left(18 - 32x_1 + 12x_1^2 + 48x_2 - 36x_1x_2 + 27x_2^2\right)\right]$				
F_{19}	$-\sum_{i=1}^{4} c_i \exp\left(-\sum_{j=1}^{3} a_{ij}\left(x_j - p_{ij}\right)^2\right)$	−3.86	F	3	$x_i \in [0, 1]$
F_{20}	$-\sum_{i=1}^{4} c_i \exp\left(-\sum_{j=1}^{6} a_{ij}\left(x_j - p_{ij}\right)^2\right)$	−3.32	F	6	$x_i \in [0, 1]$
F_{21}	$-\sum_{i=1}^{5}\left[(X-a_i)(X-a_i)^T + c_i\right]^{-1}$	−10.1532	F	4	$x_i \in [0, 10]$
F_{22}	$-\sum_{i=1}^{7}\left[(X-a_i)(X-a_i)^T + c_i\right]^{-1}$	−10.4028	F	4	$x_i \in [0, 10]$
F_{23}	$-\sum_{i=1}^{10}\left[(X-a_i)(X-a_i)^T + c_i\right]^{-1}$	−10.5363	F	4	$x_i \in [0.10]$

U: Unimodal, M: Multimodal, F: fixed-dimension multimodal

Appendix B. IEEE CEC-2017 test suite

A description of the IEEE CEC-2017 benchmark test functions is shown in Table 34.

Table 34 Characteristics of the IEEE CEC-2017 benchmark test functions

Function No.	Function	f_{min}	Range	Dim	Class
C-2017-f1	Shifted and Rotated Bent Cigar function	100	[− 100,100]	10	U
C-2017-f3	Shifted and Rotated Zakharov function	300	[− 100,100]	10	U
C-2017-f4	Shifted and Rotated Rosenbrock's function	400	[− 100,100]	10	M
C-2017-f5	Shifted and Rotated Rastrigin's function	500	[− 100,100]	10	M
C-2017-f6	Shifted and Rotated Expanded Scaffer's function	600	[− 100,100]	10	M
C-2017-f7	Shifted and Rotated Lunacek Bi-Rastrigin function	700	[− 100,100]	10	M
C-2017-f8	Shifted and Rotated Non-Continuous Rastrigin's function	800	[− 100,100]	10	M
C-2017-f9	Shifted and Rotated Levy Function	900	[− 100,100]	10	M
C-2017-f10	Shifted and Rotated Schwefel's Function	1000	[− 100,100]	10	M
C-2017-f11	Hybrid function of Zakharov, Rosenbrock and Rastrigin's	1100	[− 100,100]	10	H
C-2017-f12	Hybrid function of High Conditioned Elliptic, Modified Schwefel and Bent Cigar	1200	[− 100,100]	10	H
C-2017-f13	Hybrid function of Bent Ciagr, Rosenbrock and Lunache Bi-Rastrigin	1300	[− 100,100]	10	H
C-2017-f14	Hybrid function of Eliptic, Ackley, Schaffer and Rastrigin	1400	[− 100,100]	10	H
C-2017-f15	Hybrid function of Bent Cigar, HGBat, Rastrigin and Rosenbrock	1500	[− 100,100]	10	H
C-2017-f16	Hybrid function of Expanded Schaffer, HGBat, Rosenbrock and Modified Schwefel	1600	[− 100,100]	10	H
C-2017-f17	Hybrid function of Katsuura, Ackley, Expanded Griewank plus Rosenbrock, Modified Schwefel and Rastrigin	1700	[− 100,100]	10	H
C-2017-f18	Hybrid function of high conditioned Elliptic, Ackley, Rastrigin, HGBat and Discus	1800	[− 100,100]	10	H
C-2017-f19	Hybrid function of Bent Cigar, Rastrigin, Expanded Grienwank plus Rosenbrock, Weierstrass and expanded Schaffer	1900	[− 100,100]	10	H
C-2017-f20	Hybrid function of Happycat, Katsuura, Ackley, Rastrigin, Modified Schwefel and Schaffer	2000	[− 100,100]	10	H
C-2017-f21	Composition function of Rosenbrock, High Conditioned Elliptic and Rastrigin	2100	[− 100,100]	10	C
C-2017-f22	Composition function of Rastrigin's, Griewank's and Modifed Schwefel's	2200	[− 100,100]	10	C
C-2017-f23	Composition function of Rosenbrock, Ackley, Modified Schwefel and Rastrigin	2300	[− 100,100]	10	C
C-2017-f24	Composition function of Ackley, High Conditioned Elliptic, Girewank and Rastrigin	2400	[− 100,100]	10	C
C-2017-f25	Composition function of Rastrigin, Happycat, Ackley, Discus and Rosenbrock	2500	[− 100,100]	10	C
C-2017-f26	Composition function of Expanded Scaffer, Modified Schwefel, Griewank, Rosenbrock and Rastrigin	2600	[− 100,100]	10	C
C-2017-f27	Composition function of HGBat, Rastrigin, Modified Schwefel, Bent-Cigar, High Conditioned Elliptic and Expanded Scaffer	2700	[− 100,100]	10	C
C-2017-f28	Composition function of Ackley, Griewank, Discus, Rosenbrock, HappyCat, Expanded Scaffer	2800	[− 100,100]	10	C
C-2017-f29	Composition function of shifted and rotated Rastrigin, Expanded Scaffer and Lunacek Bi-Rastrigin	2900	[− 100,100]	10	C
C-2017-f30	Composition function of shifted and rotated Rastrigin, Non-Continuous Rastrigin and Levy function	3000	[− 100,100]	10	C

U: Unimodal, M: Multimodal, H: Hybrid, C: Composition

Appendix C. IEEE CEC-C06 2019 benchmark test functions

A description of the IEEE CEC-C06 2019 benchmark functions is given in Table 35.

Table 35 Characteristics of the IEEE CEC-C06 2019 benchmark functions

Function	Functions	Dim	Range	f_{min}
CEC01	Storn's Chebyshev polynomial fitting problem	9	[− 8192, 8192]	1
CEC02	Inverse Hilbert matrix problem	16	[− 16384, 16384]	1
CEC03	Lennard–Jones minimum energy cluster	18	[− 4, 4]	1
CEC04	Rastrigin's function	10	[− 100, 100]	1
CEC05	Grienwank's function	10	[− 100, 100]	1
CEC06	Weierstrass function	10	[− 100, 100]	1
CEC07	Modified Schwefel's function	10	[− 100, 100]	1
CEC08	Expanded schaffer's F_6 function	10	[− 100, 100]	1
CEC09	Happy CAT function	10	[− 100, 100]	1
CEC10	Ackley function	10	[− 100, 100]	1

Declarations

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

References

1. Grossmann IE, Apap RM, Calfa BA, Garcia-Herreros P, Zhang Q (2017) Mathematical programming techniques for optimization under uncertainty and their application in process systems engineering. Theor Found Chem Eng 51(6):893–909
2. Rodríguez N, Gupta A, Zabala PL, Cabrera-Guerrero G (2018) Optimization algorithms combining (meta) heuristics and mathematical programming and its application in engineering. Math Probl Eng 2018:3967457. https://doi.org/10.1155/2018/3967457
3. Harjunkoski I, Grossmann IE (2002) Decomposition techniques for multistage scheduling problems using mixed-integer and constraint programming methods. Comput Chem Eng 26(11):1533–1552
4. Chumburidze M, Basheleishvili I, Khetsuriani A (2019) Dynamic programming and greedy algorithm strategy for solving several classes of graph optimization problems. BRAIN. Broad Res Artif Intell Neurosci 10(1):101–107
5. Lan G (2020) First-order and stochastic optimization methods for machine learning. Springer, New York
6. Ommen T, Markussen WB, Elmegaard B (2014) Comparison of linear, mixed integer and non-linear programming methods in energy system dispatch modelling. Energy 74:109–118
7. Braik M, Alaa S, Al-Hiary H (2020) A novel meta-heuristic search algorithm for solving optimization problems: capuchin search algorithm. Neural Comput Appl 33(7):2515–2547
8. Braik MS (2021) Chameleon swarm algorithm: a bio-inspired optimizer for solving engineering design problems. Expert Syst Appl 174:114685
9. Mlinarić D, Perić T, Matejaš J (2019) Multi-objective programming methodology for solving economic diplomacy resource allocation problem. Croat Oper Res Rev 8:165–174
10. Sheta A, Hossam F, Braik M, Mirjalili S (2020) Nature-inspired metaheuristics search algorithms for solving the economic load dispatch problem of power system: a comparison study. Applied nature-inspired computing: algorithms and case studies. Springer, New York, pp 199–230
11. Mirjalili S, Gandomi AH, Zahra MS, Saremi S, Faris H, Mirjalili MS (2017) Salp swarm algorithm: a bio-inspired optimizer for engineering design problems. Adv Eng Softw 114:163–191
12. Sandgren E (1990) Nonlinear integer and discrete programming in mechanical design optimization 112(2):223–229
13. Qi Y, Jin L, Wang Y, Xiao L, Zhang J (2019) Complex-valued discrete-time neural dynamics for perturbed time-dependent complex quadratic programming with applications. IEEE Trans Neural Netw Learn Syst 31(9):3555–3569
14. Aktemur C, Gusseinov I (2017) A comparison of sequential quadratic programming, genetic algorithm, simulated annealing, particle swarm optimization and hybrid algorithm for the design and optimization of golinski's speed reducer. Int J Energy Appl Technol 4(2):34–52
15. Akay B, Karaboga D (2012) Artificial bee colony algorithm for large-scale problems and engineering design optimization. J Intell Manuf 23(4):1001–1014
16. Garg H (2016) A hybrid pso-ga algorithm for constrained optimization problems. Appl Math Comput 274:292–305
17. Nabil E (2016) A modified flower pollination algorithm for global optimization. Expert Syst Appl 57:192–203
18. Wang G-G (2018) Moth search algorithm: a bio-inspired metaheuristic algorithm for global optimization problems. Memet Comput 10(2):151–164
19. Li X (2003) A new intelligent optimization method-artificial fish school algorithm. Doctor thesis of Zhejiang University
20. Yang X-S (2012) Flower pollination algorithm for global optimization. International conference on unconventional computing and natural computation. Springer, New York, pp 240–249

21. Li MD, Zhao H, Weng XW, Han T (2016) A novel nature-inspired algorithm for optimization: virus colony search. Adv Eng Softw 92:65–88
22. Braik M, Sheta A (2011) Exploration of genetic algorithms and particle swarm optimization in improving the quality of medical images. In: Computational intelligence techniques in handling image processing and pattern recognition. Lambert Academic Publishing (LAP), Germany. pp 329–360
23. Arnay R, Fumero F, Sigut J (2017) Ant colony optimization-based method for optic cup segmentation in retinal images. Appl Soft Comput 52:409–417
24. Nguyen P, Kim J-M (2016) Adaptive ecg denoising using genetic algorithm-based thresholding and ensemble empirical mode decomposition. Inf Sci 373:499–511
25. Sheta A, Braik M, Al-Hiary H (2019) Modeling the tennessee eastman chemical process reactor using bio-inspired feedforward neural network (bi-ff-nn). Int J Adv Manuf Technol 103(1):1359–1380
26. Rashaideh H, Sawaie A, Al-Betar MA, Abualigah LM, Al-Laham MM, Ra'ed M, Braik M (2020) A grey wolf optimizer for text document clustering. J Intell Syst 29(1):814–830
27. Devi SG, Sabrigiriraj M (2019) A hybrid multi-objective firefly and simulated annealing based algorithm for big data classification. Concurr Comput Pract Exp 31(14):e4985
28. Mosavi MR, Khishe M, Naseri MJ, Parvizi GR, Mehdi AYAT (2019) Multi-layer perceptron neural network utilizing adaptive best-mass gravitational search algorithm to classify sonar dataset. Arch Acoust 44(1):137–151
29. Zaidan AA, Atiya B, Abu Bakar MR, Zaidan BB (2019) A new hybrid algorithm of simulated annealing and simplex downhill for solving multiple-objective aggregate production planning on fuzzy environment. Neural Comput Appl 31(6):1823–1834
30. Koppen M, Wolpert DH, Macready WG (2001) Remarks on a recent paper on the" no free lunch" theorems. IEEE Trans Evolut Comput 5(3):295–296
31. Wolpert David H, Macready William G (1997) No free lunch theorems for optimization. IEEE Trans Evol Comput 1(1):67–82
32. Yong W, Tao W, Cheng-Zhi Z, Hua-Juan H (2016) A new stochastic optimization approach-dolphin swarm optimization algorithm. Int J Comput Intell Appl 15(02):1650011
33. Boussaïd I, Lepagnot J, Siarry P (2013) A survey on optimization metaheuristics. Inf Sci 237:82–117
34. Coello Coello Carlos A (2000) Constraint-handling using an evolutionary multiobjective optimization technique. Civ Eng Syst 17(4):319–346
35. Glover F (1989) Tabu search-part i. ORSA J Comput 1(3):190–206
36. Davis L (1991) Bit-climbing, representational bias, and test suit design. Proc Intl Conf Genet Algorithm 1991:18–23
37. Lourenço HR, Martin OC, Stützle T (2003) Iterated local search. Handbook of metaheuristics. Springer, New York, pp 320–353
38. Morales-Castañeda B, Zaldivar D, Cuevas E, Fausto F, Rodríguez A (2020) A better balance in metaheuristic algorithms: does it exist? Swarm Evol Comput 54:100671
39. Yang X-S, Deb S, Fong S (2014) Metaheuristic algorithms: optimal balance of intensification and diversification. Appl Math Inf Sci 8(3):977
40. Yang X-S, Deb S, Hanne T, Xingshi H (2019) Attraction and diffusion in nature-inspired optimization algorithms. Neural Comput Appl 31(7):1987–1994
41. Yazdani M, Jolai F (2016) Lion optimization algorithm (loa): a nature-inspired metaheuristic algorithm. J Comput Des Eng 3(1):24–36
42. Kirkpatrick S, Gelatt CD, Vecchi MP (1983) Optimization by simulated annealing. Science 220(4598):671–680
43. Černỳ V (1985) Thermodynamical approach to the traveling salesman problem: an efficient simulation algorithm. J Optim Theory Appl 45(1):41–51
44. Rashedi E, Nezamabadi-Pour H, Saryazdi S (2009) Gsa: a gravitational search algorithm. Inf Sci 179(13):2232–2248
45. Mirjalili S, Mirjalili SM, Hatamlou A (2016) Multi-verse optimizer: a nature-inspired algorithm for global optimization. Neural Comput Appl 27(2):495–513
46. Hashim FA, Houssein EH, Mabrouk MS, Al-Atabany W, Mirjalili S (2019) Henry gas solubility optimization: a novel physics-based algorithm. Future Gener Comput Syst 101:646–667
47. Faramarzi A, Heidarinejad M, Stephens B, Mirjalili S (2020) Equilibrium optimizer: a novel optimization algorithm. Knowl Based Syst 191:105190
48. Rechenberg I (1973) Evolution strategy: optimization of technical systems by means of biological evolution. Fromman-Holzboog Stuttg 104:15–16
49. Goldberg David E, Holland John H (1988) Genetic algorithms and machine learning. Mach Learn 3(2):95–99
50. Holland John H (1992) Genetic algorithms. Sci Am 267(1):66–73
51. Koza John R, Koza John R (1992) Genetic programming: on the programming of computers by means of natural selection, vol 1. MIT press, Cambridge
52. Das S, Suganthan PN (2011) Differential evolution: a survey of the state-of-the-art. IEEE Trans Evol Comput 15(1):4–31
53. Eberhart R, Kennedy J (1995) A new optimizer using particle swarm theory. In: Micro Machine and Human Science, 1995. MHS'95., Proceedings of the Sixth International Symposium on, pages 39–43. IEEE
54. Colorni A, Dorigo M, Maniezzo V, et al. (1992) Distributed optimization by ant colonies. In: Proceedings of the first European conference on artificial life, volume 142, pages 134–142. Cambridge, MA
55. Karaboga D, Basturk B (2007) A powerful and efficient algorithm for numerical function optimization: artificial bee colony (ABC) algorithm. J Glob Optim 39(3):459–471
56. Saremi S, Mirjalili SZ, Mirjalili SM (2015) Evolutionary population dynamics and grey wolf optimizer. Neural Comput Appl 26(5):1257–1263
57. Mirjalili S (2016) Dragonfly algorithm: a new meta-heuristic optimization technique for solving single-objective, discrete, and multi-objective problems. Neural Comput Appl 27(4):1053–1073
58. Braik M, Sheta A, Turabieh H, Alhiary H (2020) A novel lifetime scheme for enhancing the convergence performance of salp swarm algorithm. Soft Comput 25(1):181–206
59. García-Hernández L, Salas-Morera L, Carmona-Muñoz C, Garcia-Hernandez JA, Salcedo-Sanz S (2020) A novel island model based on coral reefs optimization algorithm for solving the unequal area facility layout problem. Eng Appl Artif Intell 89:103445
60. Fausto F, Reyna-Orta A, Cuevas E, Andrade ÁG, Perez-Cisneros M (2020) From ants to whales: metaheuristics for all tastes. Artif Intell Rev 53(1):753–810
61. Loganathan GV, Geem ZW, Kim JH (2001) A new heuristic optimization algorithm: harmony search. Simulation 76(2):60–68
62. Chen W, Dai C, Zhu Y (2006) Seeker optimization algorithm. In: International Conference on Computational and Information Science, pages 167–176. Springer
63. Zhu Y, Tan Y (2010) Fireworks algorithm for optimization. In: International conference in swarm intelligence, pages 355–364. Springer
64. Lotfi S, Ramezani F (2013) Social-based algorithm (sba). Appl Soft Comput 13(5):2837–2856

65. Gandomi AH (2014) Interior search algorithm (isa): a novel approach for global optimization. ISA Trans 53(4):1168–1183
66. Ebrahimi M, Fadakar E (2016) A new metaheuristic football game inspired algorithm. In: 2016 1st conference on swarm intelligence and evolutionary computation (CSIEC), pages 6–11. IEEE
67. *Great books online*, Accessed from 17 Oct 2020. https://www.bartleby.com/16/905.html
68. Mansour W et al (2008) "Ali baba and the forty thieves": an allusion to abbasid organised crime. Glob Crime 9(1):8
69. Ma'ruf K (2015) *An analysis of diction used in the story "Ali Baba and the forty thieves" from the arabian nights written by Richard Burton and written by Marie P. Croall*. PhD thesis, IAIN Syekh Nurjati Cirebon, 2015
70. Lei Z, Gao S, Gupta S, Cheng J, Yang G (2020) An aggregative learning gravitational search algorithm with self-adaptive gravitational constants. Expert Syst Appl 152:113396
71. Wang Y, Yu Y, Gao S, Pan H, Yang G (2019) A hierarchical gravitational search algorithm with an effective gravitational constant. Swarm Evol Comput 46:118–139
72. Wang Y, Gao S, Zhou M, Yu Y (2020) A multi-layered gravitational search algorithm for function optimization and real-world problems. IEEE/CAA J Autom Sin 8(1):94–109
73. Cheng S, Shi Y, Qin Q, Zhang Q, Bai R (2014) Population diversity maintenance in brain storm optimization algorithm. J Artif Intell Soft Comput Res 4(2):83–97
74. Gao S, Yu Y, Wang Y, Wang J, Cheng J, Zhou M (2019) Chaotic local search-based differential evolution algorithms for optimization. IEEE Trans Syst Man Cybern Syst 51(6):3954–39671
75. Wang Y, Gao S, Yu Y, Wang Z, Cheng J, Yuki T (2020) A gravitational search algorithm with chaotic neural oscillators. IEEE Access 8:25938–25948
76. Jordehi AR (2015) A chaotic artificial immune system optimisation algorithm for solving global continuous optimisation problems. Neural Comput Appl 26(4):827–833
77. Bharti KK, Singh PK (2016) Opposition chaotic fitness mutation based adaptive inertia weight bpso for feature selection in text clustering. Appl Soft Comput 43:20–34
78. dos Santos Coelho L, Ayala HVH, Mariani VC (2014) A self-adaptive chaotic differential evolution algorithm using gamma distribution for unconstrained global optimization. Appl Math Comput 234:452–459
79. Liu B, Wang L, Jin Y-H, Tang F, Huang D-X (2005) Improved particle swarm optimization combined with chaos. Chaos Solitons Fractals 25(5):1261–1271
80. Jia D, Zheng G, Khan MK (2011) An effective memetic differential evolution algorithm based on chaotic local search. Inf Sci 181(15):3175–3187
81. Gao S, Vairappan C, Wang Y, Cao Q, Tang Z (2014) Gravitational search algorithm combined with chaos for unconstrained numerical optimization. Appl Math Comput 231:48–62
82. Cheng S, Shi Y, Qin Q, Gao S (2013) Solution clustering analysis in brain storm optimization algorithm. In: 2013 IEEE Symposium on Swarm Intelligence (SIS), pages 111–118. IEEE
83. Gao S, Wang Y, Wang J, Cheng JJ (2017) Understanding differential evolution: a poisson law derived from population interaction network. J Comput Sci 21:140–149
84. Wang Y, Gao S, Yu Y, Xu Z (2019) The discovery of population interaction with a power law distribution in brain storm optimization. Memet Comput 11(1):65–87
85. Kennedy J, Eberhart R (1995) Particle swarm optimization (pso). In: Proc. IEEE International Conference on Neural Networks, Perth, Australia, pages 1942–1948
86. Yan J, He W, Jiang X, Zhang Z (2017) A novel phase performance evaluation method for particle swarm optimization algorithms using velocity-based state estimation. Appl Soft Comput 57:517–525
87. Storn R, Price K (1997) Differential evolution-a simple and efficient heuristic for global optimization over continuous spaces. J Glob Optim 11(4):341–359
88. Holland JH et al (1992) Adaptation in natural and artificial systems: an introductory analysis with applications to biology, control, and artificial intelligence. MIT press, New York
89. Song Y, Wang F, Chen X (2019) An improved genetic algorithm for numerical function optimization. Appl Intell 49(5):1880–1902
90. Hansen N, Ostermeier A (1997) Convergence properties of evolution strategies with the derandomized covariance matrix adaptation: the (/i,)-es. Eufit 97:650–654
91. Hansen N, Ostermeier A (2001) Completely derandomized self-adaptation in evolution strategies. Evol Comput 9(2):159–195
92. Dorigo M, Blum C (2005) Ant colony optimization theory: a survey. Theor Comput Sci 344(2–3):243–278
93. Sayed GI, Hassanien AE, Azar AT (2019) Feature selection via a novel chaotic crow search algorithm. Neural Comput Appl 31(1):171–188
94. Civicioglu P, Besdok E (2013) A conceptual comparison of the cuckoo-search, particle swarm optimization, differential evolution and artificial bee colony algorithms. Artif Intell Rev 39(4):315–346
95. Yao X, Liu Y, Lin G (1999) Evolutionary programming made faster. IEEE Trans Evol Comput 3(2):82–102
96. Mirjalili S (2015) Moth-flame optimization algorithm: a novel nature-inspired heuristic paradigm. Knowl Based Syst 89:228–249
97. Dhiman G, Kumar V (2019) Seagull optimization algorithm: theory and its applications for large-scale industrial engineering problems. Knowl-Based Syst 165:169–196
98. Askarzadeh A (2016) A novel metaheuristic method for solving constrained engineering optimization problems: crow search algorithm. Comput Struct 169:1–12
99. Dhiman G, Kumar V (2017) Spotted hyena optimizer: a novel bio-inspired based metaheuristic technique for engineering applications. Adv Eng Softw 114:48–70
100. Mirjalili S, Mirjalili SM, Lewis A (2014) Grey wolf optimizer. Adv Eng Softw 69:46–61
101. Mirjalili S (2016) Sca: a sine cosine algorithm for solving optimization problems. Knowl-Based Syst 96:120–133
102. Bonabeau E, Marco DRDF, Dorigo M, Theraulaz G et al (1999) Swarm intelligence: from natural to artificial systems. Oxford University Press, Oxford
103. Mirjalili S, Lewis A (2016) The whale optimization algorithm. Adv Eng Softw 95:51–67
104. Zhenyu G, Bo C, Min Y, Binggang CD (2006) Self-adaptive chaos differential evolution. In: International Conference on Natural Computation, pages 972–975. Springer
105. Li-Jiang Y, Tian-Lun C (2002) Application of chaos in genetic algorithms. Commun Theor Phys 38(2):168
106. Saremi S, Mirjalili SM, Mirjalili Si (2014) Chaotic krill herd optimization algorithm. Procedia Technol 12:180–185
107. Awad Noor H, Ali Mostafa Z, Suganthan Ponnuthurai N, Reynolds Robert G (2017) Cade: a hybridization of cultural algorithm and differential evolution for numerical optimization. Inf Sci 378:215–241
108. Price KV, Awad NH, Ali MZ, Suganthan PN (2018) The 100-digit challenge: Problem definitions and evaluation criteria for the 100-digit challenge special session and competition on single objective numerical optimization. *Nanyang Technological University*

109. Abdullah JM, Ahmed T (2019) Fitness dependent optimizer: inspired by the bee swarming reproductive process. IEEE Access 7:43473–43486
110. Pereira DG, Afonso A, Medeiros FM (2015) Overview of friedman's test and post-hoc analysis. Commun Stat Simul Comput 44(10):2636–2653
111. Wang G-G, Guo L, Gandomi AH, Hao G-S, Wang H (2014) Chaotic krill herd algorithm. Inf Sci 274:17–34
112. Arora JS (2004) Optimum design concepts: optimality conditions. Introd Optim Des
113. Gandomi AH, Yang XS (2011) Benchmark problems in structural optimization. Computational optimization, methods and algorithms. Springer, New York, pp 259–281