Neural Comput & Applic
DOI 10.1007/s00521-015-1920-1

ORIGINAL ARTICLE

Dragonfly algorithm: a new meta-heuristic optimization technique for solving single-objective, discrete, and multi-objective problems

Seyedali Mirjalili[1,2]

Received: 1 October 2014 / Accepted: 30 April 2015

Abstract A novel swarm intelligence optimization technique is proposed called dragonfly algorithm (DA). The main inspiration of the DA algorithm originates from the static and dynamic swarming behaviours of dragonflies in nature. Two essential phases of optimization, exploration and exploitation, are designed by modelling the social interaction of dragonflies in navigating, searching for foods, and avoiding enemies when swarming dynamically or statistically. The paper also considers the proposal of binary and multi-objective versions of DA called binary DA (BDA) and multi-objective DA (MODA), respectively. The proposed algorithms are benchmarked by several mathematical test functions and one real case study qualitatively and quantitatively. The results of DA and BDA prove that the proposed algorithms are able to improve the initial random population for a given problem, converge towards the global optimum, and provide very competitive results compared to other well-known algorithms in the literature. The results of MODA also show that this algorithm tends to find very accurate approximations of Pareto optimal solutions with high uniform distribution for multi-objective problems. The set of designs obtained for the submarine propeller design problem demonstrate the merits of MODA in solving challenging real problems with unknown true Pareto optimal front as well. Note that the source codes of the DA, BDA, and MODA algorithms are publicly available at http://www.alimirjalili.com/DA.html.

Keywords Optimization · Multi-objective optimization · Constrained optimization · Binary optimization · Benchmark · Swarm intelligence · Evolutionary algorithms · Particle swarm optimization · Genetic algorithm

1 Introduction

Nature is full of social behaviours for performing different tasks. Although the ultimate goal of all individuals and collective behaviours is survival, creatures cooperate and interact in groups, herds, schools, colonies, and flocks for several reasons: hunting, defending, navigating, and foraging. For instance, Wolf packs own one of the most well-organized social interactions for hunting. Wolves tend to follow a social leadership to hunt preys in different steps: chasing preys, circling preys, harassing preys, and attacking preys [1, 2]. An example of collective defence is schools of fishes in oceans. Thousands of fishes create a school and avoid predators by warning each other, making the predation very difficult for predators [3]. The majority of predators have evolved to divide such schools to sub-schools by attacking them and eventually hunting the separated individuals.

Navigation is another reason for some of the creature to swarm. Birds are the best examples of such behaviours, in which they migrate between continents in flocks conveniently. It has been proven that the v-shaped configuration of flight highly saves the energy and equally distribute drag among the individuals in the flock [4]. Last but not least,

Electronic supplementary material The online version of this article (doi:10.1007/s00521-015-1920-1) contains supplementary material, which is available to authorized users.

✉ Seyedali Mirjalili
seyedali.mirjalili@griffithuni.edu.au

[1] School of Information and Communication Technology, Griffith University, Nathan Campus, Brisbane, QLD 4111, Australia

[2] Queensland Institute of Business and Technology, Mt Gravatt, Brisbane, QLD 4122, Australia

Published online: 29 May 2015

foraging is another main reason of social interactions of many species in nature. Ants and bees are the best examples of collective behaviours with the purpose of foraging. It has been proven that ants and bees are able to find and mark the shortest path from the nest/hive to the source of food [5]. They intelligently search for foods and mark the path utilizing pheromone to inform and guide others.

It is very interesting that creatures find the optimal situations and perform tasks efficiently in groups. It is obvious that they have been evolved over centuries to figure out such optimal and efficient behaviours. Therefore, it is quite reasonable that we inspire from them to solve our problems. This is then main purpose of a field of study called swarm intelligence (SI), which was first proposed by Beni and Wang in 1989 [6]. SI refers to the artificial implementation/simulation of the collective and social intelligence of a group of living creatures in nature [7]. Researchers in this field try to figure out the local rules for interactions between the individuals that yield to the social intelligence. Since there is no centralized control unit to guide the individuals, finding the simple rules between some of them can simulate the social behaviour of the whole population.

The ant colony optimization (ACO) algorithm is one of the first SI techniques mimicking the social intelligence of ants when foraging in an ant colony [8, 9]. This algorithm has been inspired from the simple fact that each ant marks its own path towards to food sources outside of the nest by pheromone. Once an ant finds a food source, it goes back to the nest and marks the path by pheromone to show the path to others. When other ants realize such pheromone marks, they also try to follow the path and leave their own pheromones. The key point here is that they might be different paths to the food source. Since a longer path takes longer time to travel for ants, however, the pheromone vaporizes with higher rate before it is re-marked by other ants. Therefore, the shortest path is achieved by simply following the path with stronger level of pheromone and abandoning the paths with weaker pheromone levels. Doringo first inspired from these simple rules and proposed the well-known ACO algorithm [10].

The particle swarm optimization (PSO) algorithm is also another well-regarded SI paradigm. This algorithm mimics the foraging and navigation behaviour of bird flocks and has been proposed by Eberhart and Kennedy [11]. The main inspiration originates from the simple rules of interactions between birds: birds tend to maintain their fly direction towards their current directions, the best location of food source obtained so far, and the best location of the food that the swarm found so far [12]. The PSO algorithm simply mimics these three rules and guides the particles towards the best optimal solutions by each of the individuals and the swarm simultaneously.

The artificial bee colony (ABC) is another recent and popular SI-based algorithm. This algorithm again simulates the social behaviour of honey bees when foraging nectar and has been proposed by Karaboga [13]. The difference of this algorithm compared to ACO and PSO is the division of the honey bees to scout, onlooker, and employed bees [14]. The employed bees are responsible for finding food sources and informing others by a special dance. In addition, onlookers watch the dances, select one of them, and follow the path towards the selected food sources. Scouters discover abandoned food sources and substitute them by new sources.

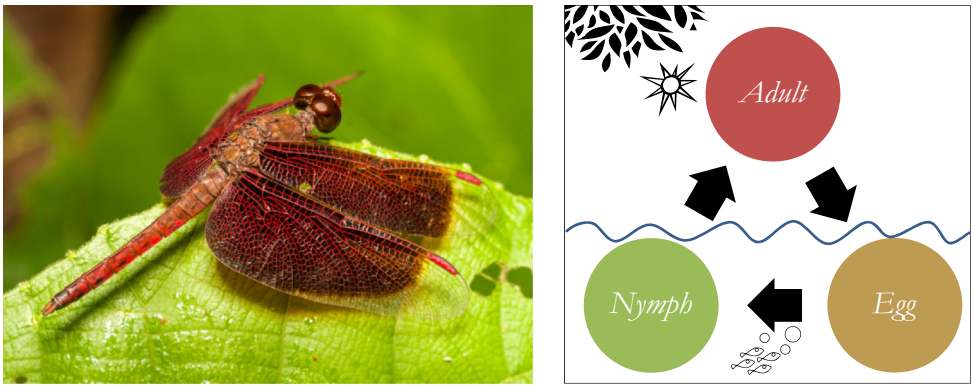
Since the proposal of these algorithms, a significant number of researchers attempted to improve or apply them in to different problems in diverse fields [15–20]. The successful application of these algorithms in science and industry evidences the merits of SI-based techniques in practice. The reasons are due to the advantages of SI-based algorithms. Firstly, SI-based techniques save information about the search space over the course of iteration, whereas such information is discarded by evolutionary algorithms (EA) generation by generation. Secondly, there are fewer controlling parameters in SI-based algorithm. Thirdly, SI-based algorithm is equipped with less operators compared to EA algorithms. Finally, SI-based techniques benefit from flexibility, which make them readily applicable to problems in different fields.

Despite the significant number of recent publications in this field [21–29], there are still other swarming behaviours in nature that have not gained deserved attention. One of the fancy insects that rarely swarm are dragonflies. Since there is no study in the literature to simulate the individual and social intelligence of dragonflies, this paper aims to first find the main characteristics of dragonflies' swarms. An algorithm is then proposed based on the identified characteristics. The no free lunch (NFL) [30] theorem also supports the motivation of this work to propose this optimizer since this algorithm may outperform other algorithms on some problems that have not been solved so far. The rest of the paper is organized as follows:

Section 2 presents the inspiration and biological foundations of the paper. The mathematical models and the DA algorithm are provided in Sect. 3. This section also proposes binary and multi-objective versions of DA. A comprehensive comparative study on several benchmark functions and one real case study is provided in Sect. 4 to confirm and verify the performances of DA, BDA, and MODA algorithms. Finally, Sect. 5 concludes the work and suggests some directions for future studies.

2 Inspiration

Dragonflies (Odonata) are fancy insects. There are nearly 3000 different species of this insect around the world [31]. As shown in Fig. 1, a dragonfly's lifecycle includes two



(a) **(b)**

Fig. 1 **a** Real dragonfly, **b** Life cycle of dragonflies (*left image* courtesy of Mehrdad Momeny at www.mehrdadmomeny.com)

main milestones: nymph and adult. They spend the major portion of their lifespan in nymph, and they undergo metamorphism to become adult [31].

Dragonflies are considered as small predators that hunt almost all other small insects in nature. Nymph dragonflies also predate on other marine insects and even small fishes. The interesting fact about dragonflies is their unique and rare swarming behaviour. Dragonflies swarm for only two purposes: hunting and migration. The former is called static (feeding) swarm, and the latter is called dynamic (migratory) swarm.

In static swarm, dragonflies make small groups and fly back and forth over a small area to hunt other flying preys such as butterflies and mosquitoes [32]. Local movements and abrupt changes in the flying path are the main characteristics of a static swarm. In dynamic swarms, however, a massive number of dragonflies make the swarm for migrating in one direction over long distances [33].

The main inspiration of the DA algorithm originates from static and dynamic swarming behaviours. These two swarming behaviours are very similar to the two main phases of optimization using meta-heuristics: exploration and exploitation. Dragonflies create sub-swarms and fly over different areas in a static swarm, which is the main objective of the exploration phase. In the static swarm, however, dragonflies fly in bigger swarms and along one direction, which is favourable in the exploitation phase. These two phases are mathematically implemented in the following section.

3 Dragonfly algorithm

3.1 Operators for exploration and exploitation

According to Reynolds, the behaviour of swarms follows three primitive principles [34]:

- Separation, which refers to the static collision avoidance of the individuals from other individuals in the neighbourhood.
- Alignment, which indicates velocity matching of individuals to that of other individuals in neighbourhood.
- Cohesion, which refers to the tendency of individuals towards the centre of the mass of the neighbourhood.

The main objective of any swarm is survival, so all of the individuals should be attracted towards food sources and distracted outward enemies. Considering these two behaviours, there are five main factors in position updating of individuals in swarms as shown in Fig. 2.

Each of these behaviours is mathematically modelled as follows:

The separation is calculated as follows [34]:

$$S_i = -\sum_{j=1}^{N} X - X_j \tag{3.1}$$

where X is the position of the current individual, X_j shows the position j-th neighbouring individual, and N is the number of neighbouring individuals.

Alignment is calculated as follows:

$$A_i = \frac{\sum_{j=1}^{N} V_j}{N} \tag{3.2}$$

where X_j shows the velocity of j-th neighbouring individual.

The cohesion is calculated as follows:

$$C_i = \frac{\sum_{j=1}^{N} X_j}{N} - X \tag{3.3}$$

where X is the position of the current individual, N is the number of neighbourhoods, and X_j shows the position j-th neighbouring individual.

Attraction towards a food source is calculated as follows:

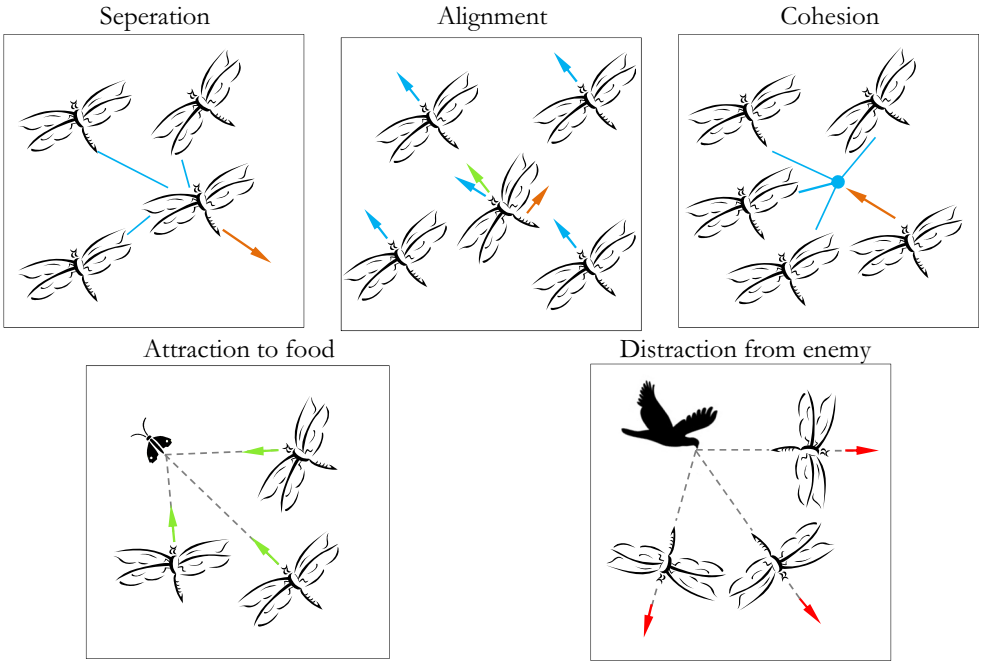


Fig. 2 Primitive corrective patterns between individuals in a swarm

$$F_i = X^+ - X \tag{3.4}$$

where X is the position of the current individual, and X^+ shows the position of the food source.

Distraction outwards an enemy is calculated as follows:

$$E_i = X^- + X \tag{3.5}$$

where X is the position of the current individual, and X^- shows the position of the enemy.

The behaviour of dragonflies is assumed to be the combination of these five corrective patterns in this paper. To update the position of artificial dragonflies in a search space and simulate their movements, two vectors are considered: step (ΔX) and position (X). The step vector is analogous to the velocity vector in PSO, and the DA algorithm is developed based on the framework of the PSO algorithm. The step vector shows the direction of the movement of the dragonflies and defined as follows (note that the position updating model of artificial dragonflies is defined in one dimension, but the introduced method can be extended to higher dimensions):

$$\Delta X_{t+1} = (sS_i + aA_i + cC_i + fF_i + eE_i) + w\Delta X_t \tag{3.6}$$

where s shows the separation weight, S_i indicates the separation of the i-th individual, a is the alignment weight, A is the alignment of i-th individual, c indicates the cohesion weight, C_i is the cohesion of the i-th individual, f is the food factor, F_i is the food source of the i-th individual, e is the enemy factor, E_i is the position of enemy of the i-th individual, w is the inertia weight, and t is the iteration counter.

After calculating the step vector, the position vectors are calculated as follows:

$$X_{t+1} = X_t + \Delta X_{t+1} \tag{3.7}$$

where t is the current iteration.

With separation, alignment, cohesion, food, and enemy factors (s, a, c, f, and e), different explorative and exploitative behaviours can achieved during optimization. Neighbours of dragonflies are very important, so a neighbourhood (circle in a 2D, sphere in a 3D space, or hypersphere in an nD space) with a certain radius is assumed around each artificial dragonfly. An example of swarming behaviour of dragonflies with increasing neighbourhood radius using the proposed mathematical model is illustrated in Fig. 3.

As discussed in the previous subsection, dragonflies only show two types of swarms: static and dynamic as shown in Fig. 4. As may be seen in this figure, dragonflies tend to align their flying while maintaining proper separation and cohesion in a dynamic swarm. In a static swarm, however, alignments are very low while cohesion is high to attack preys. Therefore, we assign dragonflies with high alignment and low cohesion weights when exploring the search space and low alignment and high cohesion when exploiting the search space. For transition between exploration and exploitation, the radii of neighbourhoods are increased proportional to the number of iterations. Another way to balance exploration and exploitation is to adaptively tune the swarming factors (s, a, c, f, e, and w) during optimization.

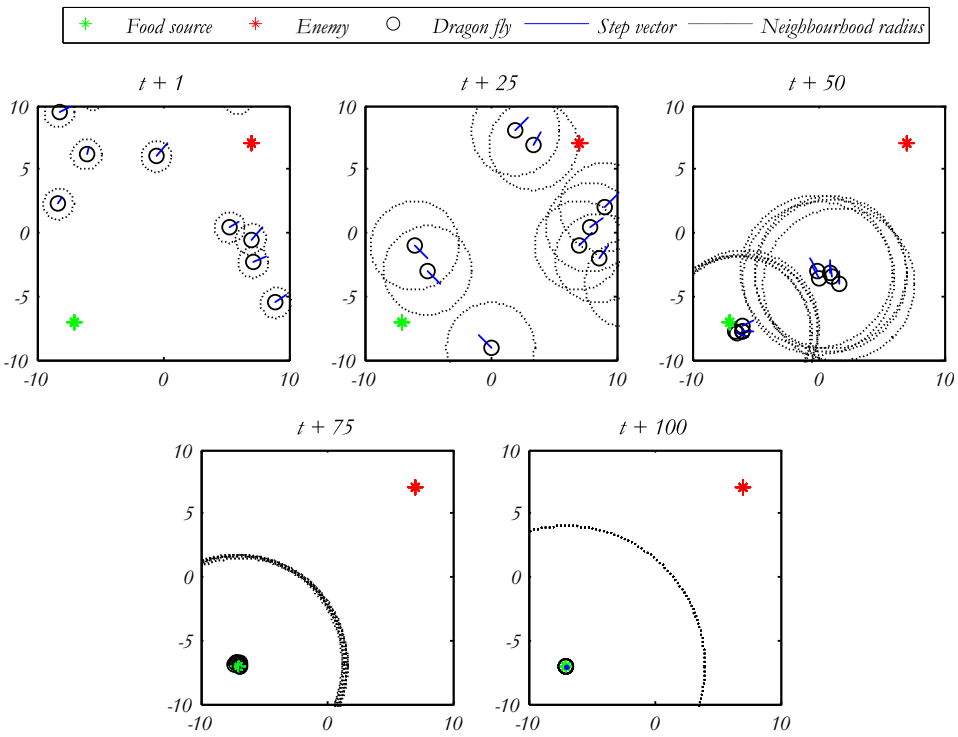


Fig. 3 Swarming behaviour of artificial dragon flies ($w = 0.9$–0.2, $s = 0.1$, $a = 0.1$, $c = 0.7$, $f = 1$, $e = 1$)

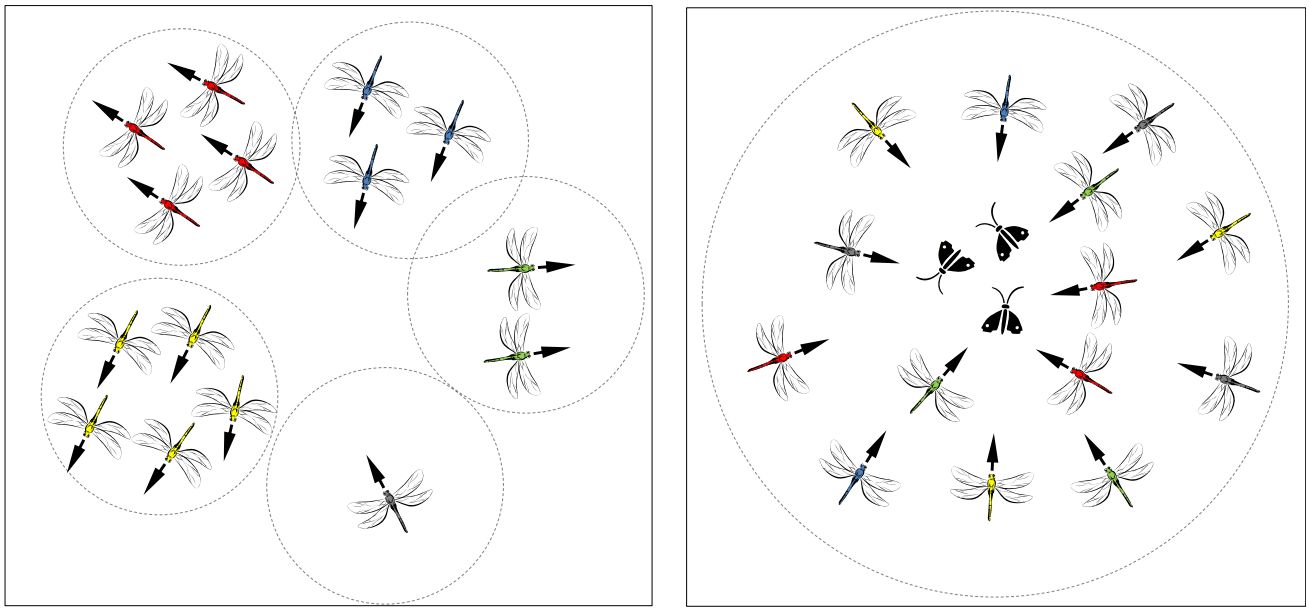


Fig. 4 Dynamic versus static dragonfly swarms

A question may rise here as to how the convergence of dragonflies is guaranteed during optimization. The dragonflies are required to change their weights adaptively for transiting from exploration to exploitation of the search space. It is also assumed that dragonflies tend to see more dragonflies to adjust flying path as optimization process progresses. In other word, the neighbourhood area is increased as well whereby the swarm become one group at the final stage of optimization to converge to the global optimum. The food source and enemy are chosen from the best and worst solutions that the whole swarm is found so far. This causes convergence towards promising areas of the search space and divergence outward non-promising regions of the search space.

To improve the randomness, stochastic behaviour, and exploration of the artificial dragonflies, they are required to fly around the search space using a random walk (Lévy flight) when there is no neighbouring solutions. In this case, the position of dragonflies is updated using the following equation:

$$X_{t+1} = X_t + \text{Lévy}(d) \times X_t \tag{3.8}$$

where t is the current iteration, and d is the dimension of the position vectors.

The Lévy flight is calculated as follows [35]:

$$\text{Lévy}(x) = 0.01 \times \frac{r_1 \times \sigma}{|r_2|^{\frac{1}{\beta}}} \tag{3.9}$$

where r_1, r_2 are two random numbers in [0,1], β is a constant (equal to 1.5 in this work), and σ is calculated as follows:

$$\sigma = \left(\frac{\Gamma(1+\beta) \times \sin\left(\frac{\pi\beta}{2}\right)}{\Gamma\left(\frac{1+\beta}{2}\right) \times \beta \times 2^{\left(\frac{\beta-1}{2}\right)}} \right)^{1/\beta} \tag{3.10}$$

where $\Gamma(x) = (x-1)!$.

3.2 The DA algorithm for single-objective problems

The DA algorithm starts optimization process by creating a set of random solutions for a given optimization problems. In fact, the position and step vectors of dragonflies are initialized by random values defined within the lower and upper bounds of the variables. In each iteration, the position and step of each dragonfly are updated using Eqs. (3.7)/(3.8) and (3.6). For updating X and ΔX vectors, neighbourhood of each dragonfly is chosen by calculating the Euclidean distance between all the dragonflies and selecting N of them. The position updating process is continued iteratively until the end criterion is satisfied. The pseudo-codes of the DA algorithm are provided in Fig. 5.

It is worth discussing here that the main differences between the DA and PSO algorithm are the consideration of separation, alignment, cohesion, attraction, distraction, and random walk in this work. Although there are some works in the literature that attempted to integrate separation, alignment, and cohesion to PSO [36–38], this paper models the swarming behaviour of dragonflies by considering all the possible factors applied to individuals in a swarm. The concepts of static and dynamic swarms are quite novel as well. The proposed model of this work is also completely different from the current improved PSO in the literature cited above.

3.3 The DA algorithm for binary problems (BDA)

Optimization in a binary search space is very different than a continuous space. In continuous search spaces, the search

Initialize the dragonflies population X_i (i = 1, 2, ..., n)
Initialize step vectors ΔX_i (i = 1, 2, ..., n)
***while** the end condition is not satisfied*
    Calculate the objective values of all dragonflies
    Update the food source and enemy
    Update w, s, a, c, f, and e
    Calculate S, A, C, F, and E using Eqs. (3.1) to (3.5)
    Update neighbouring radius
    ***if** a dragonfly has at least one neighbouring dragonfly*
        Update velocity vector using Eq. (3.6)
        Update position vector using Eq. (3.7)
    else
        Update position vector using Eq. (3.8)
    end if
    Check and correct the new positions based on the boundaries of variables
end while

Fig. 5 Pseudo-codes of the DA algorithm

agents of DA are able to update their positions by adding the step vectors to the position vectors. In a binary search space, however, the position of search agents cannot be updated by adding step vectors to X since the position vectors of search agents can only be assigned by 0 or 1. Due to the similarity of DA and other SI techniques, the current methods for solving binary problems in the literature are readily applicable to this algorithm.

According to Mirjalili and Lewis [39], the easiest and most effective method to convert a continuous SI technique to a binary algorithm without modifying the structure is to employ a transfer function. Transfer functions receive velocity (step) values as inputs and return a number in [0,1], which defines the probability of changing positions. The output of such functions is directly proportional to the value of the velocity vector. Therefore, a large value for the velocity of a search agent makes it very likely to update its position. This method simulates abrupt changes in particles with large velocity values similarly to continuous optimization (Fig. 6). Two examples of transfer functions in the literature are illustrated in Fig. 6 [39–41].

As may be seen in this figure, there are two types of transfer functions: s-shaped versus v-shaped. According to Saremi et al. [40], the v-shaped transfer functions are better than the s-shaped transfer functions because they do not force particles to take values of 0 or 1. In order to solve binary problems with the BDA algorithm, the following transfer function is utilized [39]:

$$T(\Delta x) = \left| \frac{\Delta x}{\sqrt{\Delta x^2 + 1}} \right| \tag{3.11}$$

This transfer function is first utilized to calculate the probability of changing position for all artificial dragonflies. The following new updating position formula is then employed to update the position of search agents in binary search spaces:

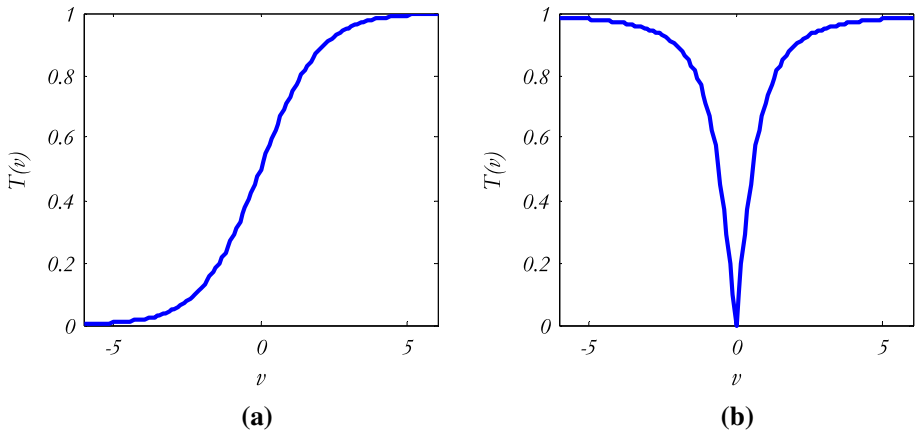


Fig. 6 *S-shaped* and *v-shaped* transfer functions

$$X_{t+1} = \begin{cases} \neg X_t & r < T(\Delta x_{t+1}) \\ X_t & r \geq T(\Delta x_{t+1}) \end{cases} \tag{3.12}$$

where r is a number in the interval of [0,1].

With the transfer function and new position updating equations, the BDA algorithm will be able to solve binary problems easily subject to proper formulation of the problem. It should be noted here that since the distance of dragonflies cannot be determined in a binary space as clearly as a continuous space, the BDA algorithm considers all of the dragonflies as one swarm and simulate exploration/exploitation by adaptively tuning the swarming factors (s, a, c, f, and e) as well as the inertia weigh (w). The pseudo-codes of the BDA algorithm are presented in Fig. 7.

Initialize the dragonflies population X_i (i = 1, 2, ..., n)
Initialize step vectors ΔX_i (i = 1, 2, ..., n)
***while** the end condition is not satisfied*
    Calculate the objective values of all dragonflies
    Update the food source and enemy
    Update w, s, a, c, f, and e
    Calculate S, A, C, F, and E using Eqs. (3.1) to (3.5)
    Update step vectors using Eq. (3.6)
    Calculate the probabilities using Eq. (3.11)
    Update position vectors using Eq. (3.12)
end while

Fig. 7 Pseudo-codes of the BDA algorithm

3.4 The DA algorithm for multi-objective problems (MODA)

Multi-objective problems have multiple objectives, which are mostly in conflict. The answer for such problems is a set of solutions called Pareto optimal solutions set. This set includes Pareto optimal solutions that represent the best trade-offs between the objectives [42]. Without loss of generality, multi-objective optimization can be formulated as a minimization problem as follows:

$$\text{Minimize}: \ F(\vec{x}) = \{f_1(\vec{x}), f_2(\vec{x}), \ldots, f_o(\vec{x})\} \tag{3.13}$$

$$\text{Subject to}: \ g_i(\vec{x}) \geq 0, \quad i = 1, 2, \ldots, m \tag{3.14}$$

$$h_i(\vec{x}) = 0, \quad i = 1, 2, \ldots, p \tag{3.15}$$

$$L_i \leq x_i \leq U_i, \quad i = 1, 2, \ldots, n \tag{3.16}$$

where o is the number of objectives, m is the number of inequality constraints, p is the number of equality constraints, and $[L_i, U_i]$ are the boundaries of i-th variable.

Due to the nature of multi-objective problems, the comparison between different solutions cannot be done by arithmetic relational operators. In this case, the concepts of Pareto optimal dominance allow us to compare two solutions in a multi-objective search space. The definitions of Pareto dominance and Pareto optimality are as follows [43]:

Definition 1 *Pareto dominance*:

Suppose that there are two vectors such as: $\vec{x} = (x_1, x_2, \ldots, x_k)$ and $\vec{y} = (y_1, y_2, \ldots, y_k)$.

Vector x dominates vector y (denote as $x \succ y$) iff:

$$\forall i \in \{1, 2, \ldots, k\}, [f(x_i) \geq f(y_i)] \wedge [\exists i \in 1, 2, \ldots, k : f(x_i)] \tag{3.17}$$

It can be seen in Eq. (3.17) that a solution dominates the other if it shows better or equal values on all objectives (dimensions) and has better value in at last one of the objectives. The definition of Pareto optimality is as follows [44]:

Definition 2 *Pareto optimality*:

A solution $\vec{x} \in X$ is called Pareto optimal iff:

$$\nexists \, \vec{y} \in X | F(\vec{y}) \succ F(\vec{x}) \tag{3.18}$$

According to the definition 2, two solutions are non-dominated with respect to each other if neither of them dominates the other. A set including all the non-dominated

solutions of a problem is called Pareto optimal set and defined as follows:

Definition 3 *Pareto optimal set*:

The set of all Pareto optimal solutions is called Pareto set as follows:

$$P_s := \{x, y \in X | \exists F(y) \succ F(x)\} \tag{3.19}$$

A set containing the corresponding objective values of Pareto optimal solutions in Pareto optimal set is called Pareto optimal front. The definition of the Pareto optimal front is as follows:

Definition 4 *Pareto optimal front*:

A set containing the value of objective functions for Pareto solutions set:

$$P_f := \{F(x) | x \in P_s\} \tag{3.20}$$

In order to solve multi-objective problems using meta-heuristics, an archive (repository) is widely used in the literature to maintain the Pareto optimal solutions during optimization. Two key points in finding a proper set of Pareto optimal solutions for a given problem are convergence and coverage. Convergence refers to the ability of a multi-objective algorithm in determining accurate approximations of Pareto optimal solutions. Coverage is the distribution of the obtained Pareto optimal solutions along the objectives. Since most of the current multi-objective algorithms in the literature are *posteriori*, the coverage and number of solutions are very important for decision making after the optimization process [45]. The ultimate goal for a multi-objective optimizer is to find the most accurate approximation of true Pareto optimal solutions (convergence) with uniform distributions (coverage) across all objectives.

For solving multi-objective problems using the DA algorithm, it is first equipped with an archive to store and retrieve the best approximations of the rue Pareto optimal solutions during optimization. The updating position of search agents is identical to that of DA, but the food sources are selected from the archive. In order to find a well-spread Pareto optimal front, a food source is chosen from the least populated region of the obtained Pareto optimal front, similarly to the multi-objective particle swarm optimization (MOPSO) algorithm [46]. To find the least populated area of the Pareto optimal front, the search space should be segmented. This is done by finding the best and worst objectives of Pareto optimal solutions obtained, defining a hyper-sphere to cover all the solutions, and dividing the hyper-spheres to equal sub-hyper-spheres in each iteration. After the creation of segments, the selection is done by a roulette-wheel mechanism with the following probability for each segment, which was proposed by Coello Coello et al. [47]:

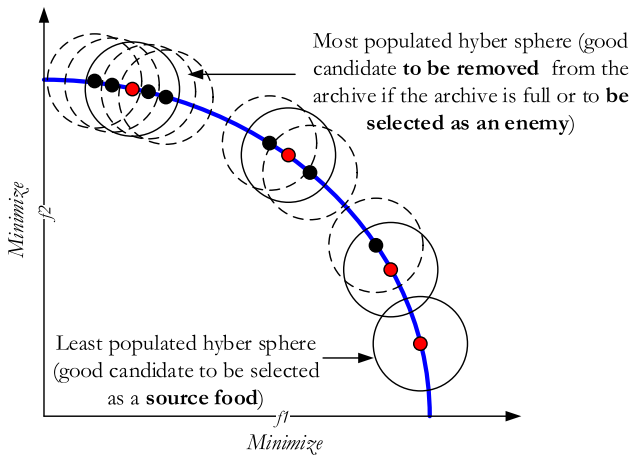


Fig. 8 Conceptual model of the best hyper-spheres for selecting a food source or removing a solution from the archive

$$P_i = \frac{c}{N_i} \tag{3.21}$$

where c is a constant number greater than one, and N_i is the number of obtained Pareto optimal solutions in the i-th segment.

This equations allows the MODA algorithm to have higher probability of choosing food sources from the less populated segments. Therefore, the artificial dragonflies will be encouraged to fly around such regions and improve the distribution of the whole Pareto optimal front.

For selecting enemies from the archive, however, the worst (most populated) hyper-sphere should be chosen in order to discourage the artificial dragonflies from searching around non-promising crowded areas. The selection is done by a roulette-wheel mechanism with the following probability for each segment:

$$P_i = \frac{N_i}{c} \tag{3.22}$$

where c is a constant number greater than one, and N_i is the number of obtained Pareto optimal solutions in the i-th segment.

In may be seen in Eq. (3.22) that the roulette-wheel mechanism assigns high probabilities to the most crowded hyper-spheres for being selected as enemies. An example of the two above-discussed selection processes is illustrated in Fig. 8. Note that the main hyper-sphere that covers all the sub-hyper-spheres is not illustrated in this figure.

The archive should be updated regularly in each iteration and may become full during optimization. Therefore, there should be a mechanism to manage the archive. If a solution is dominated by at least one of the archive residences, it should be prevented from entering the archive. If a solution dominates some of the Pareto optimal solutions in the archive, they all should be removed from

the archive, and the solution should be allowed to enter the archive. If a solution is non-dominated with respect to all of the solutions in the archive, it should be added to the archive. If the archive is full, one or more than one solutions may be removed from the most populated segments to accommodate new solution(s) in the archive. These rules are taken from the work of Coello Coello et al. [47]. Figure 8 shows the best candidate hyper-sphere (segments) to remove solutions (enemies) from in case the archive become full.

All the parameters of the MODA algorithm are identical to those of the DA algorithm except two new parameters for defining the maximum number of hyper-spheres and archive size. After all, the pseudo-codes of MODA are presented in Fig. 9.

4 Results and discussion

In this section, a number of test problems and one real case study are selected to benchmark the performance of the proposed DA, BDA, and MODA algorithms.

4.1 Results of DA algorithm

Three groups of test functions with different characteristics are selected to benchmark the performance of the DA algorithm from different perspectives. As shown in Appendix 1, the test functions are divided the three groups: unimodal, multi-modal, and composite functions [48–51]. As their names imply, unimodal test functions have single optimum, so they can benchmark the exploitation and convergence of an algorithm. In contrast, multi-modal test functions have more than one optimum, which make them more challenging than unimodal functions. One of the optima is called global optimum, and the rest are called local optima. An algorithm should avoid all the local optima to approach and approximate the global optimum. Therefore, exploration and local optima avoidance of algorithms can be benchmarked by multi-modal test functions.

The last group of test functions, composite functions, are mostly the combined, rotated, shifted, and biased version of other unimodal and multi-modal test functions [52, 53]. They mimic the difficulties of real search spaces by providing a massive number of local optima and different shapes for different regions of the search space. An algorithm should properly balance exploration and exploitation to approximate the global optimum of such test functions. Therefore, exploration and exploitation combined can be benchmarked by this group of test functions.

For verification of the results of DA, two well-known algorithms are chosen: PSO [54] as the best algorithm among swarm-based technique and GA [55] as the best evolutionary algorithm. In order to collect quantitative results, each algorithm is run on the test functions 30 times and to calculate the average and standard deviation of the best approximated solution in the last iteration. These two metrics show which algorithm behaves more stable when solving the test functions. Due to the stochastic nature of the algorithms, a statistical test is also conducted to decide about the significance of the results [56]. The averages and standard deviation only compare the overall performance of the algorithms, while a statistical test considers each run's results and proves that the results are statistically significant. The Wilcoxon non-parametric statistical test [39, 56] is conducted in this work. After all, each of the test functions is solved using

Fig. 9 Pseudo-codes of the MODA algorithm

Initialize the dragonflies population X_i *(i = 1, 2, ..., n)*
Initialize step vectors ΔX_i *(i = 1, 2, ..., n)*
Define the maximum number of hyper spheres (segments)
Define the archive size
while *the end condition is not satisfied*
    Calculate the objective values of all dragonflies
    Find the non-dominated solutions
    Update the archive with respect to the obtained non-dominated solutions
    If *the archive is full*
        Run the archive maintenance mechanism to omit one of the current archive members
        Add the new solution to the archive
    end if
    If *any of the new added solutions to the archive is located outside the hyper spheres*
        Update and re-position all of the hyper spheres to cover the new solution(s)
    end if
    Select a food source from archive: X^+*=SelectFood(archive)*
    Select an enemy from archive: X^-*=SelectEnemy(archive)*
    Update step vectors using Eq. (3.11)
    Update position vectors using Eq. (3.12)
    Check and correct the new positions based on the boundaries of variables
end while

Table 1 Statistical results of the algorithms on the test functions

Test function	DA		PSO		GA	
	Ave	Std	Ave	Std	Ave	Std
TF1	2.85E−18	7.16E−18	4.2E−18	1.31E−17	748.5972	324.9262
TF2	1.49E−05	3.76E−05	0.003154	0.009811	5.971358	1.533102
TF3	1.29E−06	2.1E−06	0.001891	0.003311	1949.003	994.2733
TF4	0.000988	0.002776	0.001748	0.002515	21.16304	2.605406
TF5	7.600558	6.786473	63.45331	80.12726	133307.1	85,007.62
TF6	4.17E−16	1.32E−15	4.36E−17	1.38E−16	563.8889	229.6997
TF7	0.010293	0.004691	0.005973	0.003583	0.166872	0.072571
TF8	−2857.58	383.6466	−7.1E+11	1.2E+12	−3407.25	164.4776
TF9	16.01883	9.479113	10.44724	7.879807	25.51886	6.66936
TF10	0.23103	0.487053	0.280137	0.601817	9.498785	1.271393
TF11	0.193354	0.073495	0.083463	0.035067	7.719959	3.62607
TF12	0.031101	0.098349	8.57E−11	2.71E−10	1858.502	5820.215
TF13	0.002197	0.004633	0.002197	0.004633	68,047.23	87,736.76
TF14	103.742	91.24364	150	135.4006	130.0991	21.32037
TF15	193.0171	80.6332	188.1951	157.2834	116.0554	19.19351
TF16	458.2962	165.3724	263.0948	187.1352	383.9184	36.60532
TF17	596.6629	171.0631	466.5429	180.9493	503.0485	35.79406
TF18	229.9515	184.6095	136.1759	160.0187	118.438	51.00183
TF19	679.588	199.4014	741.6341	206.7296	544.1018	13.30161

30 search agents over 500 iterations, and the results are presented in Tables 1 and 2. Note that the initial parameters of PSO and GA are identical to the values in the original papers cited above.

As per the results of the algorithms on the unimodal test functions (TF1–TF7), it is evident that the DA algorithm outperforms PSO and GA on the majority of the cases. The p values in Table 5 also show that this superiority is statistically significant since the p values are less than 0.05. Considering the characteristic of unimodal test functions, it can be stated that the DA algorithm benefits from high exploitation. High exploitation assists the DA algorithm to rapidly converge towards the global optimum and exploit it accurately.

The results of the algorithms on multi-modal test functions (TF8–TF13) show that again the DA algorithm provides very competitive results compared to PSO. The p values reported in Table 2 also show that the DA and PSO algorithms show significantly better results than GA. Considering the characteristics of multi-modal test functions and these results, it may be concluded that the DA algorithm has high exploration which assist it to discover the promising regions of the search space. In addition, the local optima avoidance of this algorithm is satisfactory since it is able to avoid all of the local optima and approximate the global optima on the majority of the multi-modal test functions.

Table 2 p values of the Wilcoxon ranksum test over all runs

F	DA	PSO	GA
TF1	N/A	0.045155	0.000183
TF2	N/A	0.121225	0.000183
TF3	N/A	0.003611	0.000183
TF4	N/A	0.307489	0.000183
TF5	N/A	0.10411	0.000183
TF6	0.344704	N/A	0.000183
TF7	0.021134	N/A	0.000183
TF8	0.000183	N/A	0.000183
TF9	0.364166	N/A	0.002202
TF10	N/A	0.472676	0.000183
TF11	0.001008	N/A	0.000183
TF12	0.140465	N/A	0.000183
TF13	N/A	0.79126	0.000183
TF14	N/A	0.909654	0.10411
TF15	0.025748	0.241322	N/A
TF16	0.01133	N/A	0.053903
TF17	0.088973	N/A	0.241322
TF18	0.273036	0.791337	N/A
TF19	N/A	0.472676	N/A

The results of composite test functions (TF14–TF19) show that the DA algorithm provides very competitive results and outperforms others occasionally. However,

the p values show that the superiority is not as significant as those of unimodal and multi-modal test functions. This is due to the difficulty of the composite test functions that make them challenging for algorithms employed in this work. Composite test functions benchmark the exploration and exploitation combined. Therefore, these results prove that the operators of the DA algorithm appropriately balance exploration and exploitation to handle difficulty in a challenging search space. Since the composite search spaces are highly similar to the real search spaces, these results make the DA algorithm potentially able to solve challenging optimization problems.

For further observing and analysing the performance of the proposed DA algorithm, four new metrics are employed in the following paragraphs. The main aims of this experiment is to confirm the convergence and predict the potential behaviour of the DA algorithm when solving real problems. The employed quantitative metrics are the position of dragonflies from the first to the last iteration (search history), the value of a parameter from the first to the last iteration (trajectory), the average fitness of dragonflies from the first to the last iteration, and the fitness of the best food source obtained from the first to the last iteration (convergence).

Tracking the position of dragonflies during optimization allows us to observe whether and how the DA algorithm explores and exploits the search space. Monitoring the value of a parameter during optimization assists us to observe the movement of candidate solutions. Preferably, there should be abrupt changes in the parameters in the exploration phase and gradual changes in the exploitation phase. The average fitness of dragonflies during optimization also shows the improvement in the fitness of the whole swarm during optimization. Finally, the fitness of the food source shows the improvement of the obtained global optimum during optimization.

Some of the functions (TF2, TF10, and TF17) are selected and solved by 10 search agents over 150 iterations. The results are illustrated in Figs. 10, 11, 12, and 13. Figure 10 shows the history of dragonfly's position during optimization. It may be observed that the DA algorithm tends to search the promising regions of the search space extensively. The behaviour of DA when solving TF17, which is a composite test function, is interesting because the coverage of search space seems to be high. This shows

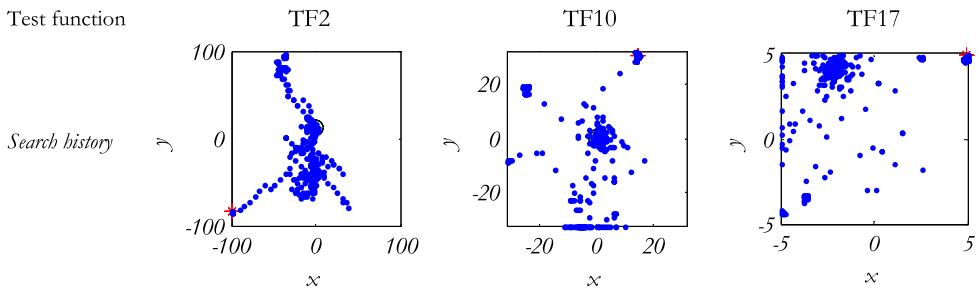


Fig. 10 Search history of the DA algorithms on unimodal, multi-modal, and composite test functions

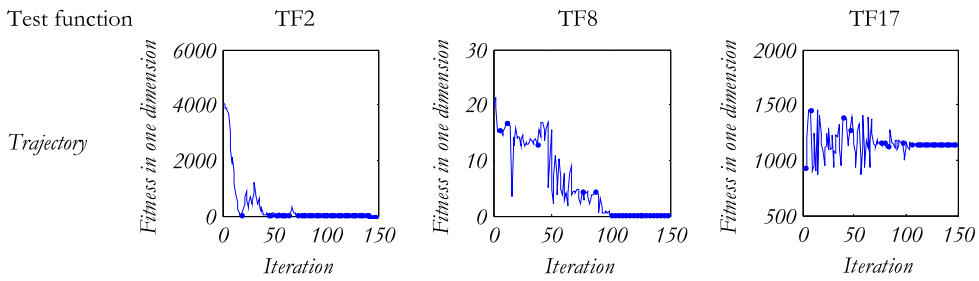


Fig. 11 Trajectory of DA's search agents on unimodal, multi-modal, and composite test functions

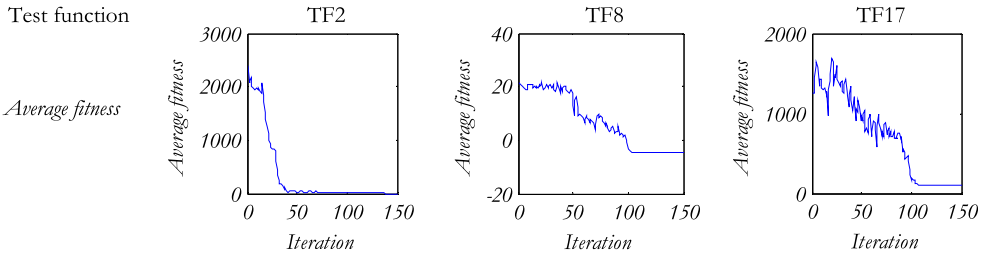


Fig. 12 Average fitness of DA's search agents on unimodal, multi-modal, and composite test functions

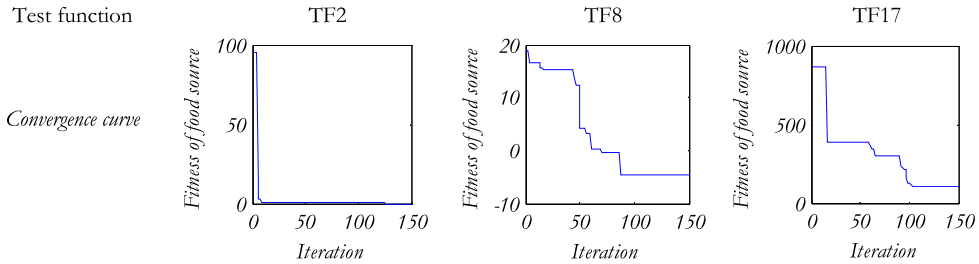


Fig. 13 Convergence *curve* of the DA algorithms on unimodal, multi-modal, and composite test functions

Table 3 Statistical results of the binary algorithms on the test functions

Test function	BDA		BPSO		BGSA	
	Ave	Std	Ave	Std	Ave	Std
TF1	0.281519	0.417723	5.589032	1.97734	82.95707	49.78105
TF2	0.058887	0.069279	0.196191	0.052809	1.192117	0.228392
TF3	14.23555	22.68806	15.51722	13.68939	455.9297	271.9785
TF4	0.247656	0.330822	1.895313	0.483579	7.366406	2.213344
TF5	23.55335	34.6822	86.44629	65.82514	3100.999	2927.557
TF6	0.095306	0.129678	6.980524	3.849114	106.8896	77.54615
TF7	0.012209	0.014622	0.011745	0.006925	0.03551	0.056549
TF8	−924.481	65.68827	−988.565	16.66224	−860.914	80.56628
TF9	1.805453	1.053829	4.834208	1.549026	10.27209	3.725984
TF10	0.388227	0.5709	2.154889	0.540556	2.786707	1.188036
TF11	0.193437	0.113621	0.47729	0.129354	0.788799	0.251103
TF12	0.149307	0.451741	0.407433	0.231344	9.526426	6.513454
TF13	0.035156	0.056508	0.306925	0.241643	2216.776	5663.491

that the DA's artificial dragonflies are able to search the search space effectively.

Figure 11 illustrates the trajectory of the first variable of the first artificial dragonfly over 150 iterations. It can be observed that there are abrupt changes in the initial iterations. These changes are decreased gradually over the course of iterations. According to Berg et al. [57], this behaviour can guarantee that an algorithm eventually convergences to a point and search locally in a search space.

Figures 12 and 13 show the average fitness of all dragonflies and the food source, respectively. The average fitness of dragonflies shows a decreasing behaviour on all of the test functions. This proves that the DA algorithm improves the overall fitness of the initial random population. A similar behaviour can be observed in the convergence curves. This also evidences that the approximation of the global optimum becomes more accurate as the iteration counter increases. Another fact that can be seen is the accelerated trend in the convergence curves. This is due to the emphasis on local search and exploitation as iteration increases which highly accelerate the convergence towards the optimum in the final steps of iterations.

As summary, the results of this section proved that the proposed DA algorithm shows high exploration and exploitation. For one, the proposed static swarm promotes exploration, assists the DA algorithm to avoid local optima, and resolves local optima stagnation when solving challenging problems. For another, the dynamic swarm of dragonflies emphasizes exploitation as iteration increases, which causes a very accurate approximation of the global optimum.

4.2 Results of BDA algorithm

To benchmark the performance of the BDA algorithm, test functions TF1 to TF13 are taken from Sect. 4.1 and Appendix 1. For simulating a binary search space, we consider 15 bits to define the variables of the test functions. The dimension of test functions is reduced from 30 to 5, so the total number of binary variables to be optimized by the BDA algorithm is 75 (5×15). For verification of the results, the binary PSO (BPSO) [58] and binary gravitational search algorithm (BGSA) [59] are chosen from the literature. Each of the algorithms is run 30 times, and the results are presented in Tables 3 and 4. Note that the initial parameters of BPSO and BGSA are identical to the values in the original papers cited above.

Table 3 shows that the proposed algorithm outperforms both BPSO and BGSA on the majority of binary test cases.

Table 4 p values of the Wilcoxon ranksum test over all runs

F	BDA	BPSO	BGSA
TF1	N/A	0.000183	0.000183
TF2	N/A	0.001706	0.000183
TF3	N/A	0.121225	0.000246
TF4	N/A	0.000211	0.000183
TF5	N/A	0.009108	0.000183
TF6	N/A	0.000183	0.000183
TF7	0.472676	N/A	0.344704
TF8	0.064022	N/A	0.000583
TF9	N/A	0.000583	0.000183
TF10	N/A	0.00033	0.00044
TF11	N/A	0.000583	0.00033
TF12	N/A	0.002827	0.000183
TF13	N/A	0.000583	0.000183

The discrepancy of the results is very evident as per the p values reported in Table 4. These results prove that the BDA algorithm inherits high exploration and exploitation from the DA algorithm due to the use of the v-shaped transfer function.

4.3 Results of MODA algorithm

As multi-objective case studies, five challenging test functions from the well-known ZDT set proposed by Deb et al. [60] are chosen in this subsection. Note that the first three test functions are identical to ZDT1, ZDT2, and ZDT3. However, this paper modifies ZDT1 and ZDT2 to have test problems with linear and tri-objective fronts as the last two case studies. The details of these test functions are available in Appendix 2. The results are collected and discussed quantitatively and qualitatively. Quantitative results are calculated by the inverse generational distance (IGD) proposed by Sierra and Coello Coello [61] over ten runs. This performance metric is similar to generational distance (GD) [62] and formulated as follows:

$$\text{IGD} = \frac{\sqrt{\sum_{i=1}^{n} d_i^2}}{n} \tag{4.1}$$

where n is the number of true Pareto optimal solutions, and d_i indicates the Euclidean distance between the i-th true Pareto optimal solution and the closest obtained Pareto optimal solutions in the reference set.

For collecting and discussing the qualitative results, the best Pareto optimal front in ten independent runs are presented. The MODA algorithm is compared to MOPSO [47] and non-dominated sorting genetic algorithm (NSGA-II) [63]. After all, the quantitative results are presented in Tables 5, 6, 7, 8, and 9, and the qualitative results are provided in Figs. 14, 15, 16, 17, and 18.

Table 5 Results of the multi-objective algorithms on ZDT1

Algorithm	IGD				
	Ave	Std	Median	Best	Worst
MODA	0.00612	0.002863	0.0072	0.0024	0.0096
MOPSO	0.00422	0.003103	0.0037	0.0015	0.0101
NSGA-II	0.05988	0.005436	0.0574	0.0546	0.0702

Table 6 Results of the multi-objective algorithms on ZDT2

Algorithm	IGD				
	Ave	Std	Median	Best	Worst
MODA	0.00398	0.001604244	0.0033	0.0023	0.006
MOPSO	0.00156	0.000174356	0.0017	0.0013	0.0017
NSGA-II	0.13972	0.026263465	0.1258	0.1148	0.1834

Table 7 Results of the multi-objective algorithms on ZDT3

Algorithm	IGD				
	Ave	Std	Median	Best	Worst
MODA	0.02794	0.004021	0.0302	0.02	0.0304
MOPSO	0.03782	0.006297	0.0362	0.0308	0.0497
NSGA-II	0.04166	0.008073	0.0403	0.0315	0.0557

Table 8 Results of the multi-objective algorithms on ZDT1 with linear front

Algorithm	IGD				
	Ave	Std	Median	Best	Worst
MODA	0.00616	0.005186	0.0038	0.0022	0.0163
MOPSO	0.00922	0.005531	0.0098	0.0012	0.0165
NSGA-II	0.08274	0.005422	0.0804	0.0773	0.0924

Table 9 Results of the multi-objective algorithms on ZDT2 with three objectives

Algorithm	IGD				
	Ave	Std	Median	Best	Worst
MODA	0.00916	0.005372	0.0063	0.0048	0.0191
MOPSO	0.02032	0.001278	0.0203	0.0189	0.0225
NSGA-II	0.0626	0.017888	0.0584	0.0371	0.0847

As per the results presented in Tables 5, 6, 7, 8, and 9, the MODA algorithm tends to outperform NSGA-II and provides very competitive results compared to MOPSO on the majority of the test functions. Figures 14, 15, 16, 17, and 18 also show that the convergence and coverage of the Pareto optimal solutions obtained by MODA algorithm are

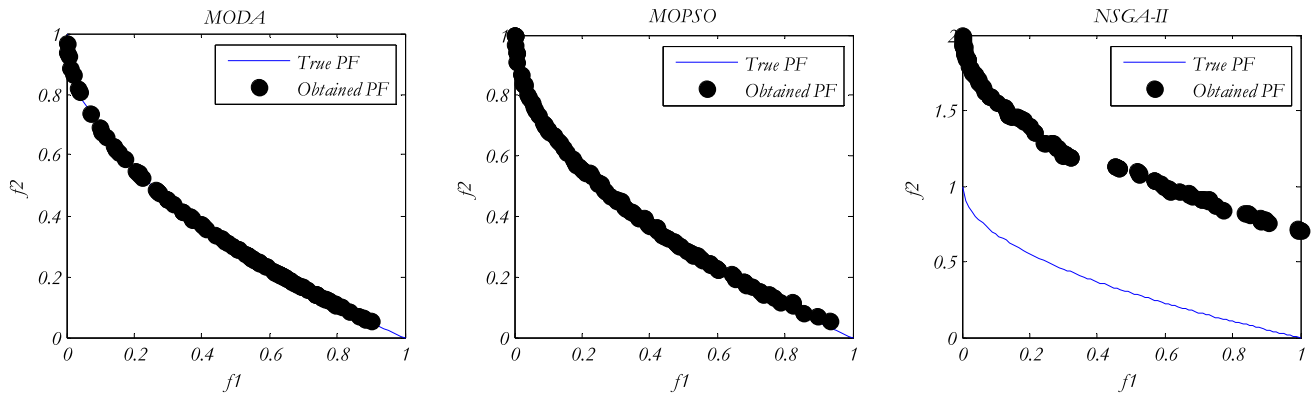


Fig. 14 Best Pareto optimal front obtained by the multi-objective algorithms on ZDT1

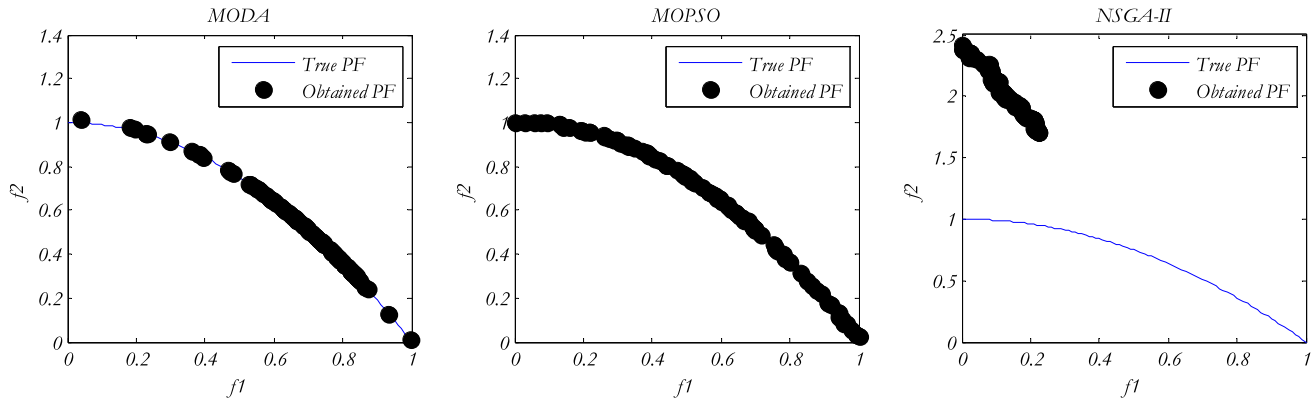


Fig. 15 Best Pareto optimal front obtained by the multi-objective algorithms on ZDT2

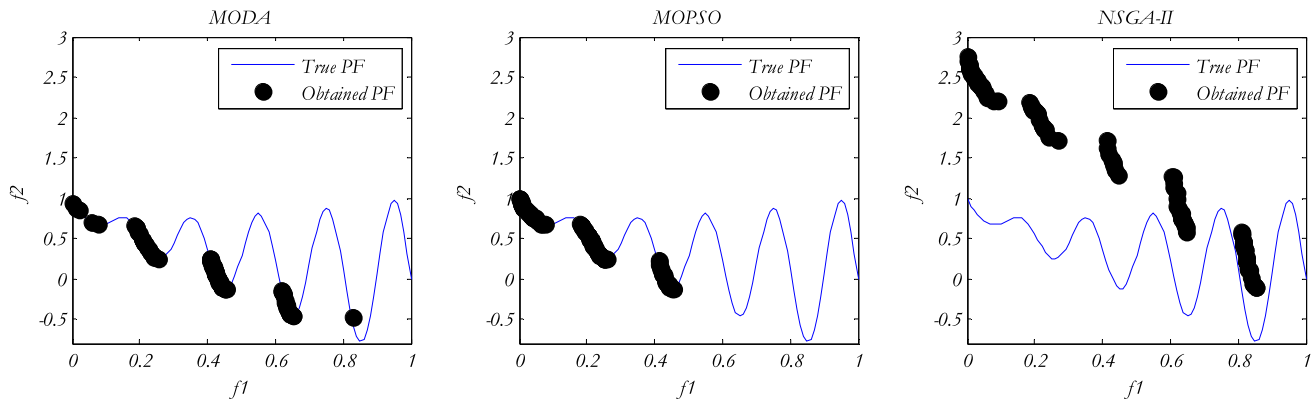


Fig. 16 Best Pareto optimal front obtained by the multi-objective algorithms on ZDT3

mostly better than NSGA-II. High convergence of the MODA originates from the accelerated convergence of search agents around the food sources selected from the archive over the course of iterations. Adaptive values for s, a, c, f, e, and w in MODA allow its search agents to converge towards the food sources proportional to the number of iterations. High coverage of the MODA algorithm is due to the employed food/enemy selection mechanisms. Since the foods and enemies are selected from the less populated and most populated hyper-spheres, respectively, the search agents of the MODA algorithm tend to search the regions of the search space that have Pareto optimal solutions with

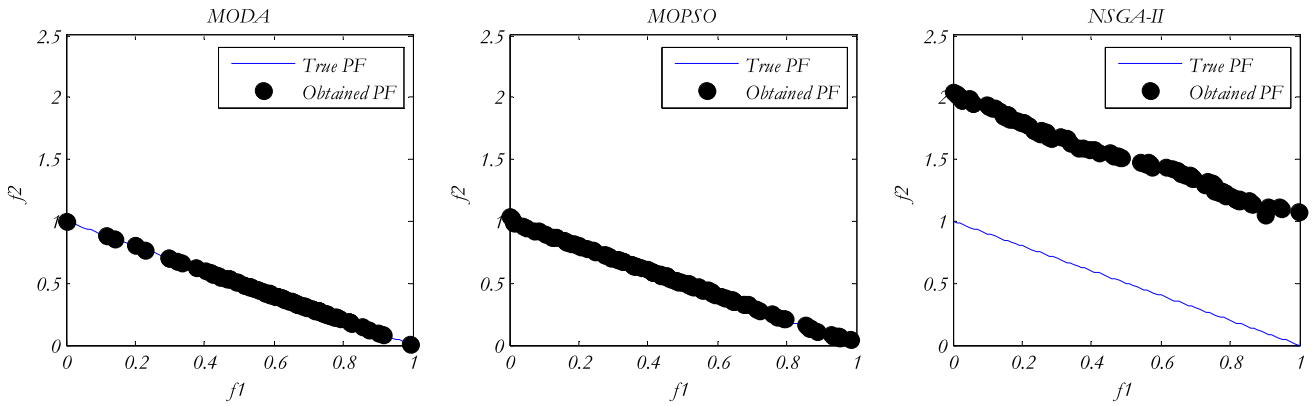


Fig. 17 Best Pareto optimal front obtained by the multi-objective algorithms on ZDT1 with linear front

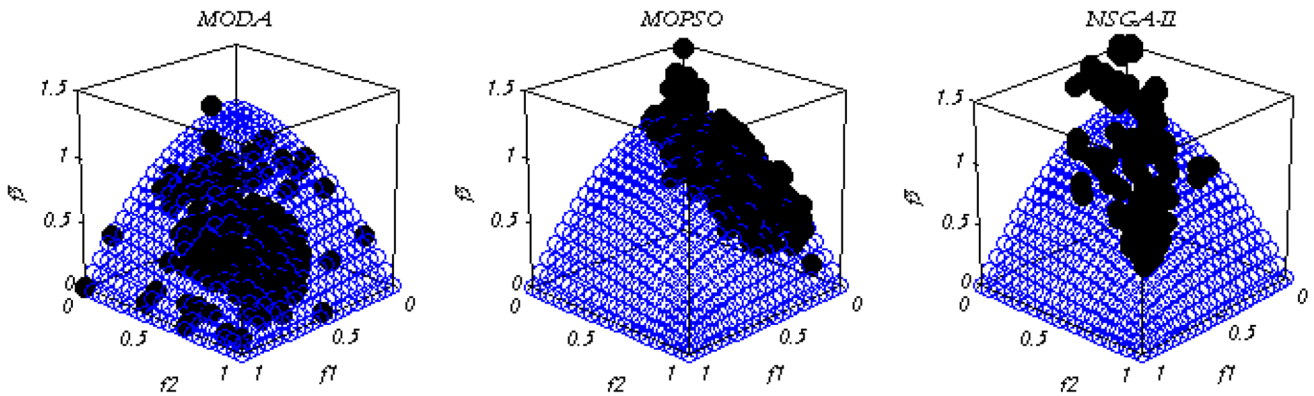


Fig. 18 Best Pareto optimal front obtained by the multi-objective algorithms on ZDT2 with three objectives

Fig. 19 A 7-blade propeller with 2 m diameter for submarines

low distribution and avoid highly distributed regions in Pareto front. Therefore, the distribution of the Pareto optimal solutions is adjusted and increased along the obtained Pareto optimal front. The maintenance mechanism for a full archive also assists the MODA algorithm to discard excess Pareto optimal solutions (enemies) in populated segments and allows adding new food sources in less populated regions. These results evidence the merits of the proposed MODA in solving multi-objective problems as a posteriori algorithm.

To demonstrate the applicability of the proposed MODA algorithm in practice, a submarine's propeller is optimized by this algorithm as well. This problem has two objectives: cavitation versus efficiency. These two objectives are in

conflict and restricted by a large number of constraints as other computational fluid dynamics (CFD) problems. This problem is formulated as follows:

$$\text{Maximize} : \eta(X) \tag{4.2}$$

$$\text{Miniimize} : V(X) \tag{4.3}$$

$$\text{Subject to} : T > 40,000,\ \text{RPM} = 200,\ Z = 7,\ D = 2,\ d = 0.4,\ \text{and}\ S = 5, \tag{4.4}$$

where η is efficiency, V is cavitation, T is thrust, RPM is rotation per second of the propeller, Z is the number of blades, D is the diameter of the propeller (m), d is the diameter of hub (m), and S is the ship speed (m/s).

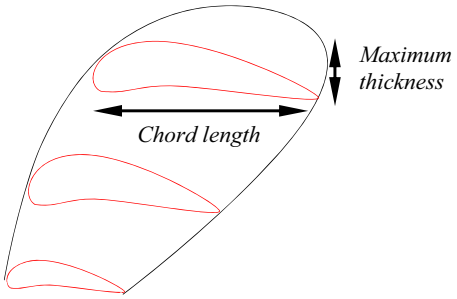


Fig. 20 A blade is divided to ten airfoils each of which has two structural parameters: maximum thickness and chord length

The shape of the propeller employed is illustrated in Fig. 19. Note that the full list of constraints and other physical details of the propeller design problem are not provided in this paper, so interested readers are referred to Carlton's book [64].

As shown in Fig. 20, the main structural parameters are the shapes of airfoils along the blades, which define the final shape of the propeller. The structure of each airfoil is determined by two parameters: maximum thickness and chord length. Ten airfoils are considered along the blade in this study, so there is a total of 20 structural parameters to be optimized by the MODA algorithm.

This real case study is solved by the MODA algorithm equipped with 200 artificial dragonflies over 300 iterations. Since the problem of submarine propeller design has many constraints, MODA should be equipped with a constraint-handling method. For simplicity, a death penalty is utilized, which assign very low efficiency and large cavitation to the artificial dragonflies that violate any of the constraints. Therefore, they are dominated automatically when finding non-dominated solutions in the next iteration.

As can be seen in Fig. 21, the MODA algorithm found 61 Pareto optimal solutions for this problem. The low density of searched points (grey dots) is due to the highly constrained nature of this problem. However, it seems that

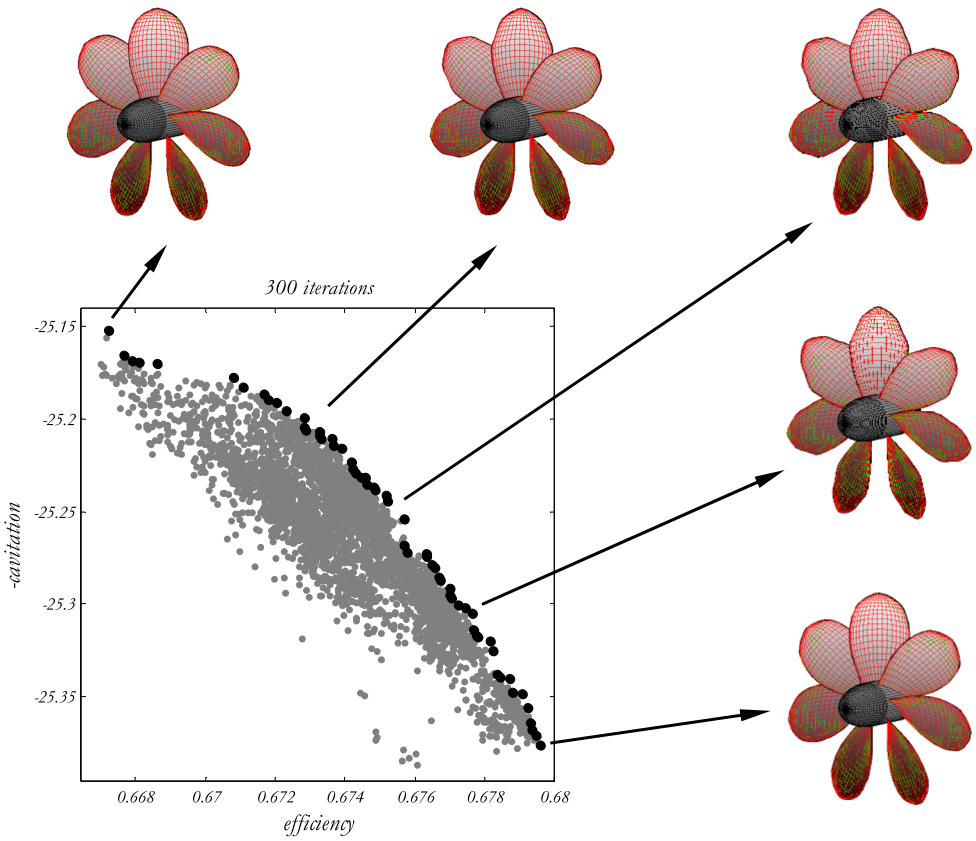


Fig. 21 Search history, obtained Pareto optimal front, and shape of some of the obtained Pareto optimal solutions by MODA

the MODA algorithm successfully improved the initial random designs and determined a very accurate approximation of the true Pareto optimal front. The solutions are highly distributed along both objectives, which confirm the coverage of this algorithm in practice as well. Therefore, these results prove the convergence and coverage of the MODA algorithm in solving real problems with unknown true Pareto optimal front. Since the propeller design problem is highly constrained, these results also evidence the merits of the proposed MODA algorithm in solving challenging constrained problems as well.

5 Conclusion

This paper proposed another SI algorithm inspired by the behaviour of dragonflies' swarms in nature. Static and dynamic swarming behaviours of dragonflies were implemented to explore and exploit the search space, respectively. The algorithm was equipped with five parameters to control cohesion, alignment, separation, attraction (towards food sources), and distraction (outwards enemies) of individuals in the swarm. Suitable operators were integrated to the proposed DA algorithm for solving binary and multi-objective problems as well. A series of continuous, binary, and multi-objective test problems were employed to benchmark the performance of the DA, BDA, and MODA algorithms from different perspectives. The results proved that all of the proposed algorithms benefits from high exploration, which is due to the proposed static swarming behaviour of dragonflies. The convergence of the artificial dragonflies towards optimal solutions in continuous, binary, and multi-objective search spaces was also observed and confirmed, which are due to the dynamic swarming pattern modelled in this paper.

The paper also considered designing a real propeller for submarines using the proposed MODA algorithm, which is a challenging and highly constrained CFD problem. The results proved the effectiveness of the multi-objective version of DA in solving real problems with unknown search spaces. As per the finding of this comprehensive study, it can be concluded that the proposed algorithms are able to outperform the current well-known and powerful algorithms in the literature. Therefore, they are recommended to researchers from different fields as open-source optimization tools. The source codes of DA, BDA, and MODA are publicly available at http://www.alimirjalili.com/DA.html.

For future works, several research directions can be recommended. Hybridizing other algorithms with DA and integrating evolutionary operators to this algorithm are two possible research avenues. For the BDA algorithm, the effects of transfer functions on the performance of this algorithm worth to be investigated. Applying other multi-objective optimization approaches (non-dominated sorting for instance) to MODA will also be valuable contributions. The DA, BDA, and MODA algorithm can all be tuned and employed to solve optimization problems in different fields as well.

Acknowledgments The author would like to thank Mehrdad Momeny for providing his outstanding dragonfly photo.

Appendix 1: Single-objective test problems utilized in this work

See Tables 10, 11, 12.

Table 10 Unimodal benchmark functions

Function	Dim	Range	Shift position	$f_{\min}$
$TF1(x) = \sum_{i=1}^{n} x_i^2$	10	[−100,100]	[−30,−30,…,−30]	0
$TF2(x) = \sum_{i=1}^{n} \lvert x_i \rvert + \prod_{i=1}^{n} \lvert x_i \rvert$	10	[−10,10]	[−3,−3,…,−3]	0
$TF3(x) = \sum_{i=1}^{n} \left(\sum_{j-1}^{i} x_j \right)^2$	10	[−100,100]	[−30,−30,…,−30]	0
$TF4(x) = \max_i \{ \lvert x_i \rvert, 1 \le i \le n \}$	10	[−100,100]	[−30,−30,…,−30]	0
$TF5(x) = \sum_{i=1}^{n-1} \left[100 \left(x_{i+1} - x_i^2 \right)^2 + (x_i - 1)^2 \right]$	10	[−30,30]	[−15,−15,…,−15]	0
$TF6(x) = \sum_{i=1}^{n} ([x_i + 0.5])^2$	10	[−100,100]	[−750,…,−750]	0
$TF7(x) = \sum_{i=1}^{n} i x_i^4 + \text{random}[0, 1)$	10	[−1.28,1.28]	[−0.25,…,−0.25]	0

Table 11 Multimodal benchmark functions

Function	Dim	Range	Shift position	$f_{\min}$
$TF8(x) = \sum_{i=1}^{n} -x_i \sin\left(\sqrt{\lvert x_i \rvert}\right)$	10	[−500, 500]	[−300,...,−300]	-418.9829×5
$TF9(x) = \sum_{i=1}^{n} \left[x_i^2 - 10\cos(2\pi x_i) + 10\right]$	10	[−5.12, 5.12]	[−2,−2,...,−2]	0
$TF10(x) = -20\exp\left(-0.2\sqrt{\frac{1}{n}\sum_{i=1}^{n} x_i^2}\right) - \exp\left(\frac{1}{n}\sum_{i=1}^{n}\cos(2\pi x_i)\right) + 20 + e$	10	[−32, 32]		0
$TF11(x) = \frac{1}{4000}\sum_{i=1}^{n} x_i^2 - \prod_{i=1}^{n}\cos\left(\frac{x_i}{\sqrt{i}}\right) + 1$	10	[−600, 600]	[−400,...,−400]	0
$TF12(x) = \frac{\pi}{n}\left\{10\sin(\pi y_1) + \sum_{i=1}^{n-1}(y_i - 1)^2\left[1 + 10\sin^2(\pi y_{i+1})\right] + (y_n - 1)^2\right\} + \sum_{i=1}^{n} u(x_i, 10, 100, 4)$ $y_i = 1 + \frac{x_i+1}{4}$ $u(x_i, a, k, m) = \begin{cases} k(x_i - a)^m & x_i > a \\ 0 & -a < x_i < a \\ k(-x_i - a)^m & x_i < -a \end{cases}$	10	[−50, 50]	[−30,−30,...,−30]	0
$TF13(x) = 0.1\left\{\sin^2(3\pi x_1) + \sum_{i=1}^{n}(x_i - 1)^2\left[1 + \sin^2(3\pi x_i + 1)\right] + (x_n - 1)^2\left[1 + \sin^2(2\pi x_n)\right]\right\} + \sum_{i=1}^{n} u(x_i, 5, 100, 4)$	10	[−50, 50]	[−100,...,−100]	0

Appendix 2: Multi-objective test problems utilized in this work

ZDT1:

$$\text{Minimise} : f_1(x) = x_1 \tag{7.1}$$

$$\text{Minimise} : f_2(x) = g(x) \times h(f_1(x), g(x)) \tag{7.2}$$

$$\text{Where} : G(x) = 1 + \frac{9}{N-1}\sum_{i=2}^{N} x_i \tag{7.3}$$

$$h(f_1(x), g(x)) = 1 - \sqrt{\frac{f_1(x)}{g(x)}} \quad 0 \le x_i \le 1, 1 \le i \le 30 \tag{7.4}$$

ZDT2:

$$\text{Minimise} : f_1(x) = x_1 \tag{7.5}$$

$$\text{Minimise} : f_2(x) = g(x) \times h(f_1(x), g(x)) \tag{7.6}$$

$$\text{Where} : G(x) = 1 + \frac{9}{N-1}\sum_{i=2}^{N} x_i \tag{7.7}$$

$$h(f_1(x), g(x)) = 1 - \left(\frac{f_1(x)}{g(x)}\right)^2 \quad 0 \le x_i \le 1, 1 \le i \le 30 \tag{7.8}$$

ZDT3:

$$\text{Minimise} : f_1(x) = x_1 \tag{7.9}$$

$$\text{Minimise} : f_2(x) = g(x) \times h(f_1(x), g(x)) \tag{7.10}$$

$$\text{Where} : G(x) = 1 + \frac{9}{29}\sum_{i=2}^{N} x_i \tag{7.11}$$

$$h(f_1(x), g(x)) = 1 - \sqrt{\frac{f_1(x)}{g(x)}} - \left(\frac{f_1(x)}{g(x)}\right)\sin(10\pi f_1(x)) \quad 0 \le x_i \le 1, 1 \le i \le 30 \tag{7.12}$$

ZDT1 with linear PF:

$$\text{Minimise} : f_1(x) = x_1 \tag{7.13}$$

$$\text{Minimise} : f_2(x) = g(x) \times h(f_1(x), g(x)) \tag{7.14}$$

$$\text{Where} : G(x) = 1 + \frac{9}{N-1}\sum_{i=2}^{N} x_i \tag{7.15}$$

$$h(f_1(x), g(x)) = 1 - \frac{f_1(x)}{g(x)} \quad 0 \le x_i \le 1, 1 \le i \le 30 \tag{7.16}$$

ZDT2 with three objectives:

$$\text{Minimise} : f_1(x) = x_1 \tag{7.17}$$

$$\text{Minimise} : f_2(x) = x_2 \tag{7.18}$$

$$\text{Minimise} : f_3(x) = g(x) \times h(f_1(x), g(x)) \times h(f_2(x), g(x)) \tag{7.19}$$

Table 12 Composite benchmark functions

Function	Dim	Range	f_{min}
TF14 (CF1)			
$f_1, f_2, f_3, \ldots, f_{10}$ = Sphere function	10	[−5, 5]	0
$[\sigma_1, \sigma_2, \sigma_3,\ldots,\sigma_{10}] = [1,1,1,\ldots1]$			
$[\lambda_1, \lambda_2, \lambda_3,\ldots,\lambda_{10}] = [5/100, 5/100, 5/100,\ldots, 5/100]$			
TF15 (CF2)			
$f_1, f_2, f_3, \ldots, f_{10}$ = Griewank's function	10	[−5, 5]	0
$[\sigma_1, \sigma_2, \sigma_3,\ldots,\sigma_{10}] = [1,1,1,\ldots1]$			
$[\lambda_1, \lambda_2, \lambda_3,\ldots,\lambda_{10}] = [5/100, 5/100, 5/100,\ldots, 5/100]$			
TF16 (CF3):			
$f_1, f_2, f_3,\ldots, f_{10}$ = Griewank's function	10	[−5, 5]	0
$[\sigma_1, \sigma_2, \sigma_3,\ldots,\sigma_{10}] = [1,1,1,\ldots1]$			
$[\lambda_1, \lambda_2, \lambda_3,\ldots,\lambda_{10}] = [1,1,1,\ldots1]$			
TF17 (CF4)			
f_1,f_2 = Ackley's function	10	[−5, 5]	0
f_3,f_4 = Rastrigin's function			
f_5,f_6 = Weierstrass function			
f_7,f_8 = Griewank's function			
f_9,f_{10} = Sphere function			
$[\sigma_1, \sigma_2, \sigma_3,\ldots,\sigma_{10}] = [1,1,1,\ldots1]$			
$[\lambda_1, \lambda_2, \lambda_3, \ldots, \lambda_{10}] = [5/32, 5/32, 1, 1, 5/0.5, 5/0.5, 5/100, 5/100, 5/100, 5/100]$			
TF18 (CF5)			
f_1,f_2 = Rastrigin's function	10	[−5, 5]	0
f_3,f_4 = Weierstrass function			
f_5,f_6 = Griewank's function			
f_7,f_8 = Ackley's function			
f_9,f_{10} = Sphere Function			
$[\sigma_1, \sigma_2, \sigma_3,\ldots,\sigma_{10}] = [1,1,1,\ldots1]$			
$[\lambda_1, \lambda_2, \lambda_3, \ldots, \lambda_{10}] = [1/5, 1/5, 5/0.5, 5/0.5, 5/100, 5/100, 5/32, 5/32, 5/100, 5/100]$			
TF19 (CF6)			
f_1,f_2 = Rastrigin's function	10	[−5, 5]	0
f_3,f_4 = Weierstrass function			
f_5,f_6 = Griewank's function			
f_7,f_8 = Ackley's function			
f_9,f_{10} = Sphere Function			
$[\sigma_1, \sigma_2, \sigma_3,\ldots,\sigma_{10}] = [0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9,1]$			
$[\lambda_1, \lambda_2, \lambda_3, \ldots, \lambda_{10}] = [0.1 \times 1/5, 0.2 \times 1/5, 0.3 \times 5/0.5, 0.4 \times 5/0.5, 0.5 \times 5/100,$ $0.6 \times 5/100, 0.7 \times 5/32, 0.8 \times 5/32, 0.9 \times 5/100, 1 \times 5/100]$			

$$\text{Where}: G(x) = 1 + \frac{9}{N-1}\sum_{i=3}^{N} x_i \tag{7.20}$$

$$h(f_1(x), g(x)) = 1 - \left(\frac{f_1(x)}{g(x)}\right)^2 \quad 0 \le x_i \le 1, 1 \le i \le 30 \tag{7.21}$$

References

1. Mirjalili S, Mirjalili SM, Lewis A (2014) Grey wolf optimizer. Adv Eng Softw 69:46–61
2. Muro C, Escobedo R, Spector L, Coppinger R (2011) Wolf-pack (*Canis lupus*) hunting strategies emerge from simple rules in computational simulations. Behav Process 88:192–197
3. Jakobsen PJ, Birkeland K, Johnsen GH (1994) Swarm location in zooplankton as an anti-predator defence mechanism. Anim Behav 47:175–178
4. Higdon J, Corrsin S (1978) Induced drag of a bird flock. Am Nat 112(986):727–744
5. Goss S, Aron S, Deneubourg J-L, Pasteels JM (1989) Self-organized shortcuts in the Argentine ant. Naturwissenschaften 76:579–581
6. Beni G, Wang J (1993) Swarm intelligence in cellular robotic systems. In: Dario P, Sandini G, Aebischer P (eds) Robots and biological systems: towards a new bionics? NATO ASI series, vol 102. Springer, Berlin, Heidelberg, pp 703–712

7. Bonabeau E, Dorigo M, Theraulaz G (1999) Swarm intelligence: from natural to artificial systems. Oxford University Press, Oxford
8. Dorigo M, Stützle T (2003) The ant colony optimization metaheuristic: algorithms, applications, and advances. In: Glover F, Kochenberger GA (eds) Handbook of metaheuristics. International series in operations research & management science, vol 57. Springer, USA, pp 250–285
9. Dorigo M, Maniezzo V, Colorni A (1996) Ant system: optimization by a colony of cooperating agents. Syst Man Cybern Part B Cybern IEEE Trans 26:29–41
10. Colorni A, Dorigo M, Maniezzo V (1991) Distributed optimization by ant colonies. In: Proceedings of the first European conference on artificial life, pp 134–142
11. Eberhart RC, Kennedy J (1995) A new optimizer using particle swarm theory. In: Proceedings of the sixth international symposium on micro machine and human science, pp 39–43
12. Eberhart RC, Shi Y (2001) Particle swarm optimization: developments, applications and resources. In: Proceedings of the 2001 congress on evolutionary computation, pp 81–86
13. Karaboga D (2005) An idea based on honey bee swarm for numerical optimization. In: Technical report-tr06, Erciyes university, engineering faculty, computer engineering department
14. Karaboga D, Basturk B (2007) A powerful and efficient algorithm for numerical function optimization: artificial bee colony (ABC) algorithm. J Global Optim 39:459–471
15. AlRashidi MR, El-Hawary ME (2009) A survey of particle swarm optimization applications in electric power systems. Evolut Comput IEEE Trans 13:913–918
16. Wei Y, Qiqiang L (2004) Survey on particle swarm optimization algorithm. Eng Sci 5:87–94
17. Chandra Mohan B, Baskaran R (2012) A survey: ant colony optimization based recent research and implementation on several engineering domain. Expert Syst Appl 39:4618–4627
18. Dorigo M, Stützle T (2010) Ant colony optimization: overview and recent advances. In: Gendreau M, Potvin J-Y (eds) Handbook of metaheuristics. International series in operations research & management science, vol 146. Springer, USA, pp 227–263
19. Karaboga D, Gorkemli B, Ozturk C, Karaboga N (2014) A comprehensive survey: artificial bee colony (ABC) algorithm and applications. Artif Intell Rev 42:21–57
20. Sonmez M (2011) Artificial Bee Colony algorithm for optimization of truss structures. Appl Soft Comput 11:2406–2418
21. Wang G, Guo L, Wang H, Duan H, Liu L, Li J (2014) Incorporating mutation scheme into krill herd algorithm for global numerical optimization. Neural Comput Appl 24:853–871
22. Wang G-G, Gandomi AH, Alavi AH (2014) Stud krill herd algorithm. Neurocomputing 128:363–370
23. Wang G-G, Gandomi AH, Alavi AH (2014) An effective krill herd algorithm with migration operator in biogeography-based optimization. Appl Math Model 38:2454–2462
24. Wang G-G, Gandomi AH, Alavi AH, Hao G-S (2014) Hybrid krill herd algorithm with differential evolution for global numerical optimization. Neural Comput Appl 25:297–308
25. Wang G-G, Gandomi AH, Zhao X, Chu HCE (2014) Hybridizing harmony search algorithm with cuckoo search for global numerical optimization. Soft Comput. doi:10.1007/s00500-014-1502-7
26. Wang G-G, Guo L, Gandomi AH, Hao G-S, Wang H (2014) Chaotic krill herd algorithm. Inf Sci 274:17–34
27. Wang G-G, Lu M, Dong Y-Q, Zhao X-J (2015) Self-adaptive extreme learning machine. Neural Comput Appl. doi:10.1007/s00521-015-1874-3
28. Mirjalili S (2015) The ant lion optimizer. Adv Eng Softw 83:80–98
29. Mirjalili S, Mirjalili SM, Hatamlou A (2015) Multi-Verse Optimizer: a nature-inspired algorithm for global optimization. Neural Comput Appl. doi:10.1007/s00521-015-1870-7
30. Wolpert DH, Macready WG (1997) No free lunch theorems for optimization. Evolut Comput IEEE Trans 1(1):67–82
31. Thorp JH, Rogers DC (2014) Thorp and Covich's freshwater invertebrates: ecology and general biology. Elsevier, Amsterdam
32. Wikelski M, Moskowitz D, Adelman JS, Cochran J, Wilcove DS, May ML (2006) Simple rules guide dragonfly migration. Biol Lett 2:325–329
33. Russell RW, May ML, Soltesz KL, Fitzpatrick JW (1998) Massive swarm migrations of dragonflies (Odonata) in eastern North America. Am Midl Nat 140:325–342
34. Reynolds CW (1987) Flocks, herds and schools: a distributed behavioral model. ACM SIGGRAPH Comput Gr 21:25–34
35. Yang X-S (2010) Nature-inspired metaheuristic algorithms, 2nd edn. Luniver Press
36. Cui Z, Shi Z (2009) Boid particle swarm optimisation. Int J Innov Comput Appl 2:77–85
37. Kadrovach BA, Lamont GB (2002) A particle swarm model for swarm-based networked sensor systems. In: Proceedings of the 2002 ACM symposium on applied computing, pp 918–924
38. Cui Z (2009) Alignment particle swarm optimization. In: Cognitive informatics, 2009. ICCI'09. 8th IEEE international conference on, pp 497–501
39. Mirjalili S, Lewis A (2013) S-shaped versus V-shaped transfer functions for binary particle swarm optimization. Swarm Evolut Comput 9:1–14
40. Saremi S, Mirjalili S, Lewis A (2014) How important is a transfer function in discrete heuristic algorithms. Neural Comput Appl:1–16
41. Mirjalili S, Wang G-G, Coelho LDS (2014) Binary optimization using hybrid particle swarm optimization and gravitational search algorithm. Neural Comput Appl 25:1423–1435
42. Mirjalili S, Lewis A (2015) Novel performance metrics for robust multi-objective optimization algorithms. Swarm Evolut Comput 21:1–23
43. Coello CAC (2009) Evolutionary multi-objective optimization: some current research trends and topics that remain to be explored. Front Comput Sci China 3:18–30
44. Ngatchou P, Zarei A, El-Sharkawi M (2005) Pareto multi objective optimization. In: Intelligent systems application to power systems, 2005. Proceedings of the 13th international conference on, pp 84–91
45. Branke J, Kaußler T, Schmeck H (2001) Guidance in evolutionary multi-objective optimization. Adv Eng Softw 32:499–507
46. Coello Coello CA, Lechuga MS (2002) MOPSO: A proposal for multiple objective particle swarm optimization. In: Evolutionary computation, 2002. CEC'02. Proceedings of the 2002 congress on, pp 1051–1056
47. Coello CAC, Pulido GT, Lechuga MS (2004) Handling multiple objectives with particle swarm optimization. Evolut Comput IEEE Trans 8:256–279
48. Yao X, Liu Y, Lin G (1999) Evolutionary programming made faster. Evolut Comput IEEE Trans 3:82–102
49. Digalakis J, Margaritis K (2001) On benchmarking functions for genetic algorithms. Int J Comput Mathematics 77:481–506
50. Molga M, Smutnicki C (2005) Test functions for optimization needs. Test functions for optimization needs. http://www.robertmarks.org/Classes/ENGR5358/Papers/functions.pdf
51. Yang X-S (2010) Test problems in optimization. arXiv preprint arXiv:1008.0549
52. Liang J, Suganthan P, Deb K (2005) Novel composition test functions for numerical global optimization. In: Swarm intelligence symposium, 2005. SIS 2005. Proceedings 2005 IEEE, pp 68–75
53. Suganthan PN, Hansen N, Liang JJ, Deb K, Chen Y, Auger A et al (2005) Problem definitions and evaluation criteria for the CEC 2005 special session on real-parameter optimization. In: KanGAL Report, vol 2005005

54. Kennedy J, Eberhart R (1995) Particle swarm optimization. In: Neural networks, 1995. Proceedings, IEEE International conference on, pp 1942–1948
55. John H (1992) Holland, adaptation in natural and artificial systems. MIT Press, Cambridge
56. Derrac J, García S, Molina D, Herrera F (2011) A practical tutorial on the use of nonparametric statistical tests as a methodology for comparing evolutionary and swarm intelligence algorithms. Swarm Evolut Comput 1:3–18
57. van den Bergh F, Engelbrecht A (2006) A study of particle swarm optimization particle trajectories. Inf Sci 176:937–971
58. J. Kennedy J, Eberhart RC (1997) A discrete binary version of the particle swarm algorithm. In: Systems, man, and cybernetics, 1997. computational cybernetics and simulation, 1997 IEEE international conference on, pp 4104–4108
59. Rashedi E, Nezamabadi-Pour H, Saryazdi S (2010) BGSA: binary gravitational search algorithm. Nat Comput 9:727–745
60. Zitzler E, Deb K, Thiele L (2000) Comparison of multiobjective evolutionary algorithms: empirical results. Evol Comput 8:173–195
61. Sierra MR, Coello Coello CA (2005) Improving PSO-based multi-objective optimization using crowding, mutation and $\in$-dominance. In: Coello Coello CA, Hernández Aguirre A, Zitzler E (eds) Evolutionary multi-criterion optimization. Lecture notes in computer science, vol 3410. Springer, Berlin, Heidelberg, pp 505–519
62. Van Veldhuizen DA, Lamont GB (1998) Multiobjective evolutionary algorithm research: a history and analysis (Final Draft) TR-98-03
63. Deb K, Pratap A, Agarwal S, Meyarivan T (2002) A fast and elitist multiobjective genetic algorithm: NSGA-II. Evolut Comput IEEE Trans 6:182–197
64. Carlton J (2012) Marine propellers and propulsion. Butterworth-Heinemann, Oxford