Applied Soft Computing Journal 93 (2020) 106392

Contents lists available at ScienceDirect

Applied Soft Computing Journal

journal homepage: www.elsevier.com/locate/asoc

Dynamic differential annealed optimization: New metaheuristic optimization algorithm for engineering applications

Hazim Nasir Ghafil [a,b,*], Károly Jármai [a]

[a] University of Miskolc, Miskolc, H-3515 Miskolc, Egyetemváros, Hungary
[b] Faculty of Engineering, University of Kufa, Najaf, Iraq

ARTICLE INFO

Article history:
Received 24 October 2019
Received in revised form 23 April 2020
Accepted 7 May 2020
Available online 15 May 2020

Keywords:
Dynamic differential annealed optimization
Optimization algorithms
Computational intelligence
Path planning
Engineering design optimization

ABSTRACT

This work proposes a novel optimization algorithm which can be used to solve a wide range of mathematical optimization problems where the global minimum or maximum is required. The new algorithm is based on random search and classical simulated annealing algorithm (it mimics the modern process of producing high-quality steel) and is designated dynamic differential annealed optimization (DDAO). The proposed algorithm was benchmarked for 51 test functions. The dynamic differential annealed optimization algorithm has been compared to a large number of highly cited optimization algorithms. Over numerical tests, DDAO has outperformed some of these algorithms in many cases and shown high performance. Constrained path planning and spring design problem were selected as a practical engineering optimization problem. DDAO converged to the global minimum of problems efficiently, and for spring design problem DDAO has found the best feasible solution than what is found by many algorithms.

1. Introduction

Metaheuristics have been developed and widely used in different disciplines due to their efficiency, reliability and relatively low computing time. Several natural and human-made phenomena have been inspiring researchers to develop new metaheuristics for mathematical and engineering optimization. One reason is that these phenomena represent predefined logical plans for the developers to build an algorithm. Optimization algorithms have been utilized for a wide spectrum of applications [1] in many branches of sciences and have become the best choice to solve complicated problems where analytical solutions are difficult to find. Metaheuristics can be classified into different categories [2] according to their base inspiration to the following:

1.1. Physics and human-based algorithms

Simulated annealing algorithm [3] was inspired by the annealing process where the molecular structures changes during cooling iron from high temperature to low temperature, resulting in better properties for the annealed steel. Thermal exchange optimization [4] was developed based on Newton's law of cooling, which is stating that the rate of cooling is proportional the difference between the temperature of the body and temperature of the atmosphere. The improvisation of a musical note in the jazz trio was the base idea of the harmony search [5]. Gravitational search algorithm (GSA) [6] is an optimization based on the law of gravity and mass interactions. Multi-Verse optimizer (MVO) [7] mimics the three concepts in cosmology which are white hole, black hole, and wormhole. A Sine Cosine Algorithm (SCA) [8] generates random initial candidate solutions based on the sine and cosine functions. Other physical operations have been used to develop algorithms like Equilibrium [9], movement of electrons through the orbits around the nucleus of an atom [10].

1.2. Natural inspired algorithms

One example is the behaviour of viruses [11], where the virus adopts the diffusion and mechanism of the host cell to form its strategy for infection. The mechanism of flower pollination –the answer to why flowering plants have dominated the landscape for millions of years – inspired the flower pollination algorithm [12]. Another example of inspiration is the behaviour of bark beetles [13] and an artificial immune system [14] that led to the development of powerful optimizers. Ant lion optimization (ALO) [15] has been inspired by the hunting mechanism of antlions in nature. Another version of ALO was developed for multi-objective optimization and solving engineering problems called (MOALO) [16]. Grey wolf optimization (GWO) [17] with another version called chaotic grey wolf optimization (CGWO) [18]

* Corresponding author at: University of Miskolc, Miskolc, H-3515 Miskolc, Egyetemváros, Hungary.
E-mail address: vegyhnr@uni-miskolc.hu (H.N. Ghafil).

https://doi.org/10.1016/j.asoc.2020.106392

mimics the hunting mechanism of grey wolves in nature. GWO is an interesting work and has different versions in the literature like Multi-objective grey wolf optimizer (MOGWO) [19]. The inspiration of the dragonfly algorithm (DA) [20] has come from the static and dynamic behaviours of dragonflies. Harris hawks optimization (HHO) [21] was inspired by the surprise pounce, which is a chasing style of Harris' hawks. Moth-flame optimization algorithm (MFO) [22] is a nature-inspired optimization and developed based on the navigation method of moths. The Whale Optimization Algorithm (WOA) [23] mimics the social behaviour of humpback whales. Many other things occurring in nature have inspired algorithms like farmland fertility [24], human dynasties [25], barnacles mating [26], Marine Predators [27], Poor and rich [28], Slime mould [2], fruit fly [29].

1.3. Swarm optimization algorithms

The strategy of ants, while they search for the shortest path between a food source and their colony, gave the inspiration for ant colony optimization [30]. Spider monkey optimization Algorithm (SMO) [31] mimics Fission–Fusion social (FFS) structure of spider monkeys and well replicate the fundamentals of swarm intelligence which are self-organization and division of labour. Forging behaviour is always a matter of inspiration, and in this context, we can mention the artificial bee colony algorithm (ABC) [32]. Particle swarm optimization (PSO) [33,34]depicts sharing knowledge in societies of birds or fishes. Other societies have inspired swarm optimization algorithms like tunicate swarm [35], Salp Swarm [36], Grasshoppers [37], collective movement of the animal group [38].

1.4. Evolutionary algorithms

One of the earlier works to mimic a phenomenon to find the global minimum or maximum of an optimization problem was done in 1975 by Holland [39] when he introduced his genetic algorithm (GA), which mimics the theory of evolution by Charles Darwin. Several years later, GA was improved by the differentiation strategy, and the result was the differential evolution algorithm [40,41]. Bayesian optimization algorithm (BOA) [42] is based on genetic algorithms. Also, (CMA-ES) [43] which is an evolutionary optimization strategy was developed to reduce the complexity of the derandomized evolution strategy with covariance matrix adaptation. There are many other evolutionary algorithms which are using the same principle of evolution to find optimal solutions like Find-Fix-Finish-Exploit-Analyze [44], evolutionary population dynamics and grasshopper optimization [45].

There many metaheuristic algorithms that depict natural or artificial behaviour as well as hybrid algorithms which can be a combination of two different metaheuristics from different categories. Forging behaviour is always a matter of inspiration, and in this context, we can mention the artificial bee colony algorithm (ABC) [32]. ABC algorithm is well-known swarm optimization algorithm and many works were written to improve its performance or reduce its drawbacks like improving its efficiency by hybridization with differential evolution algorithm [46]. Optimization algorithms have been utilized for a wide spectrum of applications [1] in many branches of sciences and have become the best choice to solve complicated problems where analytical solutions are difficult to find. The simplest form of optimization tool is random search [47], but there are many optimization algorithms, each of which is suitable for a certain category of problems. In summary, there is no algorithm able to solve all problems [48]. This work proposes a novel optimization algorithm based on simulated annealing [3,49]. The new optimization algorithm was developed to solve a wide range of optimization problems in different applications of artificial intelligence like path and trajectory planning [50,51], robotic kinematic synthesis [52], solving nonlinear applications [53], linear and nonlinear equations [54], structural optimization [55,56] and any problem where the global minimum or maximum is desired.

The new algorithm was inspired by the process of improving the quality of industrial steel to produce dual-phase steel [57,58]. This human-made phenomenon was followed to implement a mathematical model which is the heart of the proposed optimizer. The new algorithm shares some aspects of random search, differential evolution, and simulated annealing and will be denoted as dynamic differential annealed optimization. DDAO has been extensively tested using 51 benchmarks. The proposed algorithm was compared with several algorithms mentioned in state-of-the-art optimizers using different benchmarks: Firefly, flower pollination, particle swarm optimization [59], invasive weeds optimization, harmony search algorithm [60], GWO, WOA, and others mentioned in respective tables. In some cases, dynamic differential annealed optimization outperformed these algorithms. Practical engineering problems dealing with path planning [61,62] and spring design also have been efficiently solved with DDAO. The simulation results of the constrained optimization of engineering problems have been compared with state-of-art algorithms discussed in the relevant sections. The importance of the study is related to the optimization of engineering design, where in many cases we search for the minimum or maximum of a quantity rather than the stability of the algorithm itself. For example, estimating the minimum thickness of an object or maximum load which a member can withstand in engineering applications is what we care about, and how the algorithm converges is not the issue. In many experiments during this work, DDAO was found to be superior in finding the minimum of a function compared to other meta-heuristics. In the steel industry, the annealing process gives the iron better properties like increasing flexibility, and this operation has given the inspiration for the simulated annealing algorithm (SA) [3,49]. However, the annealing process is an old technique, and nowadays, there are many recent techniques to produce superior steel like dual-phase steel (DP) which has better properties hundred times than annealed steel. Up to our knowledge, there is no optimization algorithm mimics or simulates the production of the superior dual-phase steel. Furthermore, production of the dual-phase steel is totally different than the annealing process even when they look similar, and this difference reflects the difference between SA and DDAO algorithms. The proposed algorithm is a chaotic and independent on the population size, and we have used fixed three members as a population in all our experiments even when we have compared its efficiency with other algorithms that have 30 members in their population. As a result, DDAO is essentially different than differential evolution algorithms which they are dependent on their population size.

2. Related works

In the context of this work, we think that it is better to review and express the similarities and differences between DDAO algorithm and other related works which are differential evolution (DE) and simulated annealing (SA). Also, this section shows the overall changes in the factors of simulated annealing.

2.1. Differential evolution

Despite the similarity, the main difference between the genetic algorithm (GA) [5] and differential evolution (DE) is the mutation process and selection [41]. In genetic algorithm, the solutions with better fitness are highly possible to be chosen as

parents while in DE all the solutions have the same chance to be selected as parents. DE chooses the solutions that provide better convergence while GA chooses solutions even if they do not have significant advantage to the convergence. That is the main reason why DE is faster than GA in the convergence performance. The main steps of DE are:

Initialization
Evaluation
Repeat
Mutation
Recombination
Evaluation
Selection
Until (termination criteria are met)

This section is developed to prevent any misunderstanding in Section 3 and to cite the essential differences between DE and DDAO algorithm.

2.2. Simulated annealing

In 1983, the simulated annealing algorithm [49] was developed based on the Monte Carlo model [63] to simulate the effect of the energy levels on the molecular structure of the solids. The set of annealing schedule is the cooling factor, stopping and starting temperature, and the number of moves at each temperature [64]. The following represents the SA algorithm:

Define loop: for T=Tmax to Tmin,
Initialize random current solution C within the search space,
Find the value of the current solution Ec by the objective function,
Randomly create neighbour solution N,
Find the value of the neighbour solution En by the objective function,
Estimate the difference between objectives values of the current solution and neighbour solution $\Delta E = Ec - EN$,
if $\Delta E > 0$,
$C = N$,
elseif $e^{\frac{\Delta E}{T}}$ > *uniformly distributed random number [0,1], this is the acceptance criteria,*
$C = N$,
end until stop criterion,

where T is a variable starts with high value (Tmax) and ends with low value (Tmin) and stop criterion can be defined as a maximum number of iteration or reaching minimum temperature (Tmin). SA and GA were combined [65] to get the benefit of SA on local search and the benefit of GA on a global search. Also, the combined effect of SA and Tabo search TS was employed to develop new hybridization [66]. Population-based simulated annealing (PSA)was developed to enhance the drawbacks of the SA algorithm [67], PSA uses the population's ability to explore the search space. Four search techniques; Simulated Annealing (SA) and Threshold Annealing (TA) algorithms; Golden Ratio space search strategy and Markovian Model are run in parallel to execute hybrid SA algorithm [68]. Particle swarm optimization and SA were hybridized to develop swarm-based simulated annealing [69]. An improved simulated annealing algorithm based on old Japanese techniques [70] has introduced two other factors to the original SA algorithm, folding and reheating. Under this approach, folding is conceived as a compression of the search space, while the reheating is a reinitialization of the cooling process in the original SA scheme. SA algorithm was improved by merging linear programming as an intensification [71]. Also, SA was used as an upper-level and differential evolution as lower-level to develop an algorithm for heat exchanger network synthesis [72]. Thus, in most of the cases, SA was improved by combining it with another optimization algorithm.

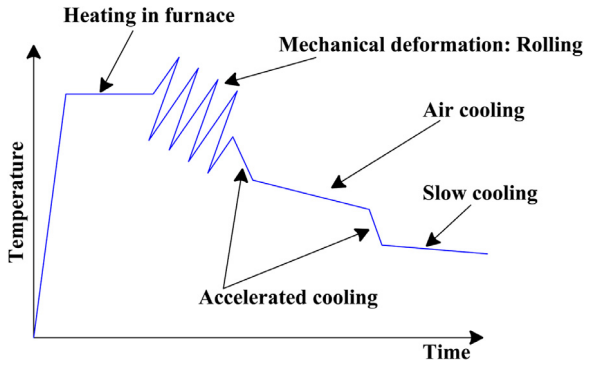

Fig. 1. Description of dual-phase steel production process.

3. Dynamic differential annealed optimization

Many studies have been conducted to improve the mechanical properties of steel [73,74] in order to reduce the required mass for the manufacturing process [75]; this means a reduction in the cost [76]. These studies deal with developing high strength steels, with considerable formation rates, designated as advanced high strength steels [77]. Dual-phase steel [58] is one type of advanced high strength steels with a unique microstructure that gives its super mechanical properties. Producing dual-phase steel requires an accurately controlled annealing system combined with deformation of the steel component [78]. Fig. 1 illustrates a general scheme describing the operation of improving the microstructure of ordinary steel to become dual-phase steel with ultra mechanical properties. In general, at an elevated temperature iron melts to a liquid state and by continuous cooling, it converts to a solid-state at room temperature. The properties of the metal depend on the current phases in the solid-state: for instance, the ferritic phase gives ductility, the martensite phase is responsible for hardness, and there are other phases like bainite and austenite. Usually, many iron phases exist together in the same solid, and they give a combination of effects to the final mechanical properties of the steel. To produce dual-phase steel (DP), a piece of iron is heated to a high temperature above the recrystallization point, where the microstructure is approximately homogeneous. After a period, the piece of iron has to be rolled to a thinner thickness and during rolling the metal will cool down. After applying mechanical deformation, the metal is subjected to subsequent differential cooling; accelerated cooling, air cooling, another accelerated cooling, and finally slow cooling to room temperature. This process leads to steel with a microstructure of ferritic matrix containing a hard martensitic phase in the form of islands. The combined effects of the soft ferrite and hard martensite give dual-phase steel high strength with an acceptable formation rate, which is very important, especially in the automotive industry [79].

The previously described systematic procedure to produce a high-quality steel type (dual-phase) is a clear process of optimization. The metal gets improved from a low-quality to a high-quality condition. This behaviour inspired the dynamic differential annealed optimization algorithm developed in this study. DDAO can be mathematically formulated as follows:

1. The mass of steel, initially, consists of groups of molecules (the population of candidate solutions) that should be improved to be one mixture of ferrite and martensite (the global solution).

2. In dual-phase steel manufacturing, the temperature decreases, and, for each reduction in temperature, there is a chance to form a different phase of steel. This is equivalent to the iteration process in mathematical optimization while searching for a global solution.

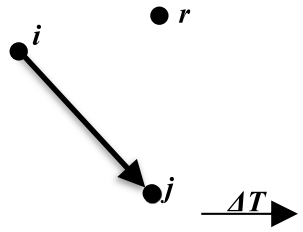

Fig. 2. Different energy levels are different candidate solutions in the search space.

3. Each phase in metal has its internal energy, and this is equivalent to the value of the objective function in mathematical optimization.

4. Fig. 1 shows different cooling rates, air cooling, accelerated cooling, and slow cooling; we have proposed the following equation to represent the cooling operation

$$S^k = (Sc_i - Sc_j) + Sr, \tag{1}$$

where S^k is a new solution proposed for the iteration number (k), $k = 1\ldots n$ where n is the number of iterations, and Sc_i and Sc_j, are randomly chosen solutions from the population with random (i) and (j) indices. Sr is a randomly generated solution within the search space of the problem out of the population. For more explanation, consider Fig. 2, where the temperature is changed by ΔT from point i to point j. We have mentioned that each energy level is equivalent to a solution with objective value, in other words, the difference between two random energy levels is also a solution. Further, adding this difference to a random energy in the space, point r in Fig. 2, will return a new value of energy which is a solution with different energy (objective value). This an imaginary basis for Eq. (1), which is the backbone of the search engine of the proposed algorithm and it is responsible for the main convergence for optimization problems. It is worth, mentioning that Eq. (1) and its complementary equation (3) are similar to the mutation process mentioned in differential evolution [41], but in our case there is no evolution, there is random selection, and even Sr is chosen randomly out of the population from another sub-population, as we will explain later.

5. During the differential reduction of the heat, the metal is rolled, and this mechanical operation must be mathematically modelled. For programming reasons, we should assume that the metal undergoes forging instead of rolling. The dynamic behaviour of the hammer during forging can be represented as a parameter fluctuating between 1 and a random number

$$f = \begin{cases} 1 & if\ rem(iteration, 2) = 1 \\ random\ [0, 1] & if\ rem(iteration, 2) = 0 \end{cases}, \tag{2}$$

where f is the forging parameter and rem is the remainder after division on 2. During iterations, Eq. (2) means that when the current number of the iteration is odd i.e. 1, 3, 5.etc. f will be one while if the current number of iteration is even (2,4,6…etc.) f will be a random number between zero and one. This is consistent with forging in real life shown in Fig. 3 when the applied force by hummer fluctuates between no change and some random value. Since forging is done online with differential cooling, Eq. (1) is modified as:

$$S^k = (Sc_i - Sc_j) + Sr * f. \tag{3}$$

6. In the real annealing process, it is more likely to accept the formation of new phases at elevated temperatures than at low temperatures. In the optimization process, we suggest the same procedure depending on the probability formula described by SA algorithm [49]:

$$P = e^{\frac{-\Delta E}{T}}, \tag{4}$$

$$\Delta E = \frac{Cost(S^k) - Cost(S_L)}{Cost(S_L)}, \tag{5}$$

where P is the probability of accepting a new solution, ΔE is the difference between the objective value of the proposed solution from Eq. (3) and the objective value of the solution S_L, which is a solution of index L in the population, $L = 1,\ldots$, population size. T is the temperature variable, which should start with high value and be updated during iterations constantly to a lower value. The proposed solution can be accepted if $P >$ random number $\in [0, 1]$. At the beginning of the search, T starts with high value; consequently, P will be close to one, according to Eq. (4). This means that a wide range of random numbers can be less that one and the solution will be selected. At the low value of T, the probability P will be close to zero; according to Eq. (4), this means that a very narrow range of random numbers could be less than P and the solution is less likely to be selected. For example, $e^{-1/300} = 0.9967$while $e^{-1/0.3} = 0.0357$, this is a simple mechanism to reduce the probability of selection of a new solution as the temperature is reduced.

7. The process is repeated from step 4, and the best solution is stored for each iteration.

The main search engine is random search while dynamic annealing process is a correction on the search; that is why it is named dynamic differential annealed optimization. A MATLAB platform has been used to implement dynamic differential annealed optimization (DDAO). Fig. 4 shows the pseudocode of the algorithm, while Fig. 5 presents a flowchart.

The new parameter; forging parameter f have a noticeable impact on the overall performance of DDAO during optimizing mathematical problems. For some set of problems, DDAO is better

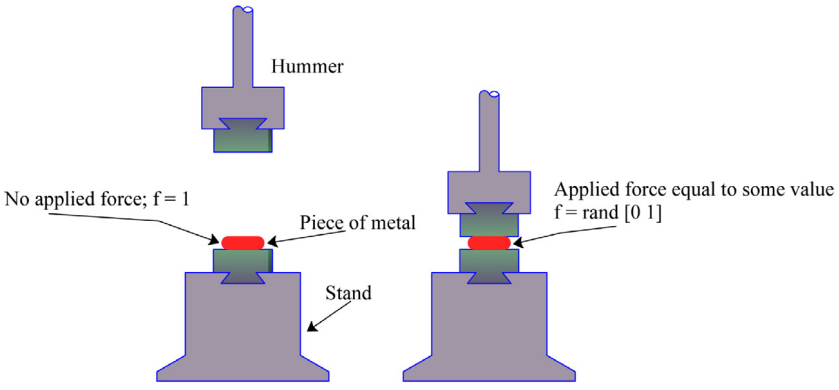

Fig. 3. Forging operation.

```
Initialize population Xi (i=1,2,…,n)
Initialise parameter T, cooling rate
Calculate the cost of each solution
Xb= The best solution
While (t<Max iteration)
Initialise sub-population S
Calculate the cost of the sub-population
Sort sub-population
Sr = Best solution in sub-population
Choose two random solutions Xm and Xn from population
Calculate Sk from equation (3)
Sort population X
   foreach solution in population X
     if there is an improvement
        Xi = Sk
        otherwise, replace the worst solution in population X using equations (4) and (5)
     endif
   endfor
Update Xb
T=T*cooling rate
t=t+1
endwhile
return Xb
```

Fig. 4. Pseudocode of proposed DDAO algorithm.

Table 1
CEC 2015 Expensive benchmark problems.

CEC 2015			
Type	No.	Description	*fmin*
Unimodal functions	F1	Rotated Bent Cigar Function	100
	F2	Rotated Discus Function	200
Simple multimodal functions	F3	Shifted and Rotated Weierstrass Function	300
	F4	Shifted and Rotated Schwefel's Function	400
	F5	Shifted and Rotated Katsuura Function	500
	F6	Shifted and Rotated HappyCat Function	600
	F7	Shifted and Rotated HGBat Function	700
	F8	Shifted and Rotated Expanded Griewank's plus Rosenbrock's Function	800
	F9	Shifted and Rotated Expanded Scaffer's F6 Function	900
Hybrid functions	F10	Hybrid Function 1 (N = 3)	1000
	F11	Hybrid Function 2 (N = 4)	1100
	F12	Hybrid Function 3 (N = 5)	1200
Composition Functions	F13	Composition Function 1 (N = 5)	1300
	F14	Composition Function 2 (N = 3)	1400
	F15	Composition Function 3 (N = 5)	1500

Table 2
Unimodal benchmark functions.

No.	Function	*d*	*Range*	*fmin*
F16	$f=\sum_{i=1}^{n} x_i^2$	30	[−100,100]	0
F17	$f=\sum_{i=1}^{n} \lvert x_i\rvert + \prod_{i=1}^{n} \lvert x_i\rvert$	30	[−100,100]	0
F18	$f=\sum_{i=1}^{n} \left(\sum_{j-1}^{i} x_j\right)^2$	30	[−100,100]	0
F19	$f=\max_i \{\lvert x_i\rvert, 1\le i\le n\}$	30	[−100,100]	0
F20	$f=\sum_{i=1}^{n-1} \left(100\left(x_{i+1}-x_i^2\right)^2+(x_i-1)^2\right)$	30		
F21	$f=\sum_{i=1}^{n} ([x_i+0.5])^2$	30	[−100,100]	0
F22	$f=\sum_{i=1}^{n} ix_i^4 + random[0,1)$	30	[−1.28,1.28]	0

with f equal to 1 in Eq. (3), worse with f equal to a random value, and vis versa for another set of problems. Thus, equation (2) gives a solution for this problem when half of the iterations consider $f=1$ and the rest of the iteration consider *f= random* [0 1]. We have presented this issue with numerical results in Section 7.4. However, forging parameter should have more attention for future work. DDAO has a straightforward structure with three parameters; the maximum number of iterations, number of sub iterations, and cooling rate that can be adjusted. While DDAO is independent of the population size, a fixed population can be used for future work. While DDAO has excellent exploration performance, researchers can increase the performance of the algorithm in future by increasing the exploitation. The source codes of this algorithm can be found in https://www.mathworks.com/matlabcentral/fileexchange/75526-dynamic-differential-annealed-optimization-ddao.

4. Results and discussion

In this section, the DDAO algorithm is benchmarked on 51 benchmark functions. DDAO algorithm has two function evaluations for each iteration and not fair to compare its results

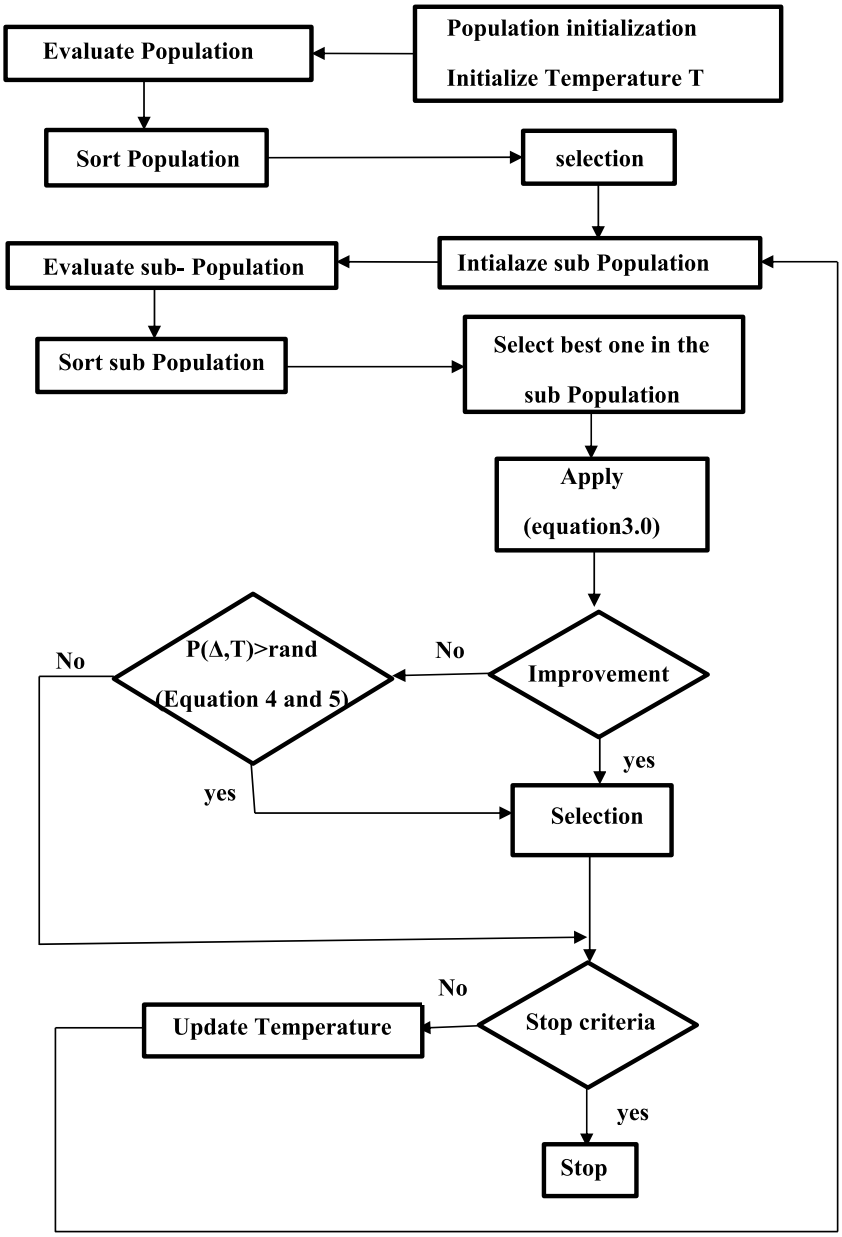

Fig. 5. DDAO flow chart.

with other metaheuristics in terms of the number of iterations. Thus, in all our experiments, we have used a specific amount of function evaluations to have a fair comparison with other published statistical results. The only exception is that we have used a specific interval of time to do the comparative experiments in Section 4.2. In all the experiments, DDAO has performed the results with population size equal to three and cooling rate of 0.995. The maximum number of iterations and the maximum number of sub iterations were modified to have the proper function evaluations matching FEs of other metaheuristics for the sake of fair comparisons. Any change in the internal parameter of DDAO will be mentioned in the respective section.

4.1. CEC 2015

CEC 2015 special session [80] benchmarks are listed in Table 1. CEC 2015 functions are divided into four groups: unimodal, simple unimodal, hybrid, and composed test functions where *fmin* is the global minimum of the functions. The DDAO algorithm was run 20 times on CEC 2015 benchmarks, and the maximum number of function evaluations was 1500 for 30-dimensional problems (30D). We have compared DDAO results with other statistical results [81] used to compare hybrid Firefly and particle swarm optimization algorithm (HFPSO), particle swarm optimization (PSO) [33], FA [82], and hybrid firefly and particle optimization algorithm (FFPSO) [83]. The DDAO algorithm has to evaluate a test function 1500 time per each 20 independent runs as the same as the run condition of HFPSO to have a fair comparison, and the results are reported in Table 8. The results are in terms of mean which is the average value and STD which is the standard deviation.

4.2. Time-based mathematical optimization

In this experiment, we have employed 23 benchmarks which are utilized by many researchers [17] and listed in Tables 2–4 where *fmin* is the optimum, *d* is the variable size, and *range*

Table 3
Multimodal benchmark functions.

No.	Function	d	Range	fmin
F23	$f=\sum_{i=1}^{n}-x_i\sin\left(\sqrt{\lvert x_i\rvert}\right)$	30	[−500,500]	−418.9829 × 5
F24	$f=\sum_{i=1}^{n}\left[x_i^2-10\cos\left(2\pi x_i\right)+10\right]$	30	[−5.12,5.12]	0
F25	$f=-20\exp\left(-20\sqrt{\frac{1}{n}\sum_{i=1}^{n}x_i^2}\right)-\exp\left(\frac{1}{n}\sum_{i=1}^{n}\cos\left(2\pi x_i\right)\right)+20+e$	30	[−32,32]	0
F26	$f=\frac{1}{4000}\sum_{i=1}^{n}x_i^2-\Pi_i^n\cos\left(\frac{x_i}{\sqrt{i}}\right)+1$	30	[−600,600]	0
F27	$f=\frac{\pi}{n}\left\{10\sin\left(\pi y_1\right)+\sum_{i=1}^{n}\left(y_i-1\right)^2\left[1+10\sin 2\left(\pi y_{i+1}\right)\right]+\left(y_n-1\right)^2\right\}+\sum_{i=1}^{n}u\left(x_i,10,100,4\right)$ $y_i=1+\frac{x_i+1}{4}$ $u(x_i,u,k,m)=\begin{cases}k(x_i-a)^m x_i>a\\0-a<x_i<a\\k(-x_i-a)^m x_i<-a\end{cases}$	30	[−50,50]	0
F28	$f=0.1\left\{\sin 2\left(3\pi x_1\right)+\sum_{i=1}^{n}\left(x_i-1\right)^2\left[1+\sin 2\left(3\pi x_i+1\right)\right]+\left(x_n-1\right)^2\left[1+\sin 2\left(2\pi x_n\right)\right]\right\}+$ $\sum_{i=1}^{n}u\left(x_i,5,100,4\right)$	30	[−50,50]	0
F29	$f=\left(\frac{1}{500}+\sum_{j=1}^{25}\frac{1}{j+\sum_{i-1}^{2}\left(x_i-a_{ij}\right)^2}\right)^{-1}$	2	[−65,65]	1
F30	$f=\left[e^{-\sum_{i=1}^{n}\left(x_i/\beta\right)^{2m}}-2e^{\sum_{i=1}^{n}x_i^2}\right].\Pi_{i=1}^{n}\cos 2x_i,\ m=5$	30	[−20, 20]	−1
F31	$f=\left\{\left(\sum_{i=1}^{n}\sin 2\left(x_i\right)\right)-\exp\left(-\sum_{i=1}^{n}x_i^2\right)\right\}.\exp\left(-\sum_{i=1}^{n}\sin 2\sqrt{\lvert x_i\rvert}\right)$	30	[−10,10]	−1

Table 4
Fixed-dimension multimodal benchmark functions.

No.	Function	d	Range	fmin
F32	$f=\left(x_2-\frac{5.1}{4\pi^2}x_1^2+\frac{5}{\pi}x_1-6\right)^2+10\left(1-\frac{1}{8\pi}\right)\cos x_1+10$	2	[−5,5]	0.398
F33	$f=\left[1+\left(x_1+x_2+1\right)^2\left(19-14x_1+3x_1^2-14x_2+6x_1x_2+3x_2^2\right)\right]\times$ $\left[30+\left(2x_1-3x_2\right)^2\times\left(18-32x_1+12x_1^2+48x_2-36x_1x_2+27x_2^2\right)\right]$	2	[−2,2]	3
F34	$f=-\sum_{i=1}^{4}c_i\exp\left(-\sum_{i=1}^{3}a_{ij}\left(x_i-p_{ij}\right)^2\right)$	3	[1,3]	−3.86
F35	$f=-\sum_{i=1}^{4}C_i\exp\left(-\sum_{j=1}^{6}a_{ij}\left(x_j-p_{ij}\right)^2\right)$	6	[0,1]	3.32
F36	$f=-\sum_{i=1}^{5}\left[\left(x-a_i\right)\left(x-a_i\right)^T+c_i\right]^{-1}$	4	[0,10]	−10.1532
F37	$f=-\sum_{i=1}^{7}\left[\left(x-a_i\right)\left(x-a_i\right)^T+c_i\right]$	4	[0,10]	
F38	$f=-\sum_{i=1}^{10}\left[\left(x-a_i\right)\left(x-a_i\right)^T+c_i\right]^{-1}$	4	[0,10]	10.5363

is the limits of the search space. These test functions are divided into three main groups: unimodal, multimodal, and fixed dimensions. DDAO was compared with ALO, HHO, BOA, DA, GWO, PSO, SCA, and WOA. In this experiment, a maximum amount of elapsed time 0.5 s for each 51 independent runs is used to examine the performance of the algorithms. DACA algorithm uses double FE for each run with three population size while the rest of the algorithms use single FE with 30 population size. However, this is fair comparison because, in the end, each algorithm in this experiment has only 0.5 s to optimize the benchmark and show the results. All the source codes for these algorithms are found at https://www.mathworks.com/matlabcentral/fileexchange/ . and the statistical results are reported in Tables 9–11.

where $a=\begin{bmatrix}4&4&4&4\\1&1&1&1\\8&8&8&8\\6&6&6&6\\3&7&3&7\\2&9&2&9\\5&5&3&3\\8&1&8&1\\6&2&6&2\\7&3.6&7&3.6\end{bmatrix}$ and $c^T=\begin{bmatrix}0.1\\0.2\\0.2\\0.4\\0.4\\0.6\\0.3\\0.7\\0.5\\0.5\end{bmatrix}$

Table 5
Rastrigin, De Jong 1, and Schwefel functions.

Function	Name	Expression	*Range*	F_{min}
F39	Rastrigin	$f(x)=\sum_{i=1}^{n}\left(x_i^2-10\cos(2\pi.x_i)+10\right)$	[−5.12, 5.12]	0
F40	De Jong1	$f(x)=\sum_{i=1}^{n}\left(x_i^2\right)$	[−5.12, 5.12]	0
F41	Schwefel 2.22	$f(x)=\sum_{i=1}^{n}\lvert x_i\rvert+\prod_{i=1}^{n}\lvert x_i\rvert$	[−10, 10]	0

Table 6
Small-scale benchmarks [84].

Function	Name	Expression	*d*	*Range*	*Fmin*
F42	Bohachevsky	$f(x)=x_1^2+2x_2^2-0.3\cos(3\pi.x_1)-0.4\cos(4\pi.x_2)+0.7$	2	[−100, 100]	0
F43	Booth	$f(x)=(x_1+2x_2-7)^2+(2x_1+x_2-5)^2$	2	[−10, 10]	0
F44	Matyas	$f(x)=0.26\left(x_1^2+x_2^2\right)-0.48x_1.x_2$	2	[−10, 10]	0
F45	Easom	$f(x)=-\cos(x_1).\cos(x_2).\exp\left(-(x_1-\pi)^2-(x_2-\pi)\right)$	2	[−100, 100]	−1
F46	Beale	$f(x)=(1.5-x_1+x_1x_2)^2+\left(2.25-x_1+x_1x_2^2\right)^2+\left(2.625-x_1+x_1x_2^3\right)^2$	2	[−4.5, 4.5]	0
F47	Schaffer N.2	$f(x)=0.5+\frac{\sin 2\left(\sqrt{x_1^2+x_2^2}\right)-0.5}{\left(1+0.001\left(x_1^2+x_2^2\right)\right)^2}$	2	[−100, 100]	0
F48	Six-hump Camel	$f(x)=4x_1^2-2.1x_1^4+\frac{1}{3}x_1^6+x_1x_2+\left(-4+4x_2^2\right)x_2^2$	2	[−2, 2]	−1.03163

Table 7
Small-scale benchmarks [84].

Function	Name	Expression	*d*	*Range*	*Fmin*
F49	Drop-Wave	$f(x)=-\frac{1+\cos\left(12\sqrt{x_1^2+x_2^2}\right)}{0.5\left(x_1^2+x_2^2\right)+2}$	2	[−5.12,5.12]	−1
F50	Eggholder	$f(x)=-(x_2+47)\sin\left(\sqrt{\left\lvert x_2+\frac{x_1}{2}+47\right\rvert}\right)-x_1\sin\left(\sqrt{\lvert x_1-(x_2+47)\rvert}\right)$	2	[−512,512]	−959.6407
F51	Holder table	$f(x)=-\left\lvert\sin(x_1)\cos(x_2)\exp\left(\left\lvert1-\frac{\sqrt{x_1^2+x_2^2}}{\pi}\right\rvert\right)\right\rvert$	2	[−10, 10]	−19.2085
F52	Levy N.13	$f(x)=\sin 2(3\pi x_1)+(x_1-1)^2\left[1+\sin 2(3\pi x_2)\right]+(x_2-1)^2\left[1+\sin 2(3\pi x_2)\right]$	2	[−10, 10]	0

Table 8
Statistical results of DDAO algorithm on CEC 2015 for 30D.

F	PSO		FA		FFPSO		HPSOFF		HFPSO		DDAO	
	Mean	Std.	Mean	Std.	Mean	Std.	Mean	Std.	Mean	Std.	Mean	Std.
F1	3.9049E+09	1.6017E+09	2.8899E+10	5.8485E+09	9.3283E+10	1.2799E+10	4.7539E+09	1.3400E+09	1.1795E+09	6.2432E+08	**9.6046E+08**	**3.5041E+08**
F2	9.9760E+04	2.5812E+04	1.3418E+05	2.6344E+04	6.9430E+06	1.4507E+07	9.7376E+04	2.0814E+04	8.5653E+04	2.0924E+04	6.8922E+10	7.5671E+09
F3	3.3113E+02	2.8347E+00	3.3850E+02	1.9604E+00	3.4771E+02	1.9232E+00	3.3059E+02	2.6174E+00	3.2638E+02	4.1408E+00	**3.2113E+02**	**6.2131E−02**
F4	7.7928E+03	5.2384E+02	8.0330E+03	4.2608E+02	9.6696E+03	4.0262E+02	6.8199E+03	8.5746E+02	5.1202E+03	6.7938E+02	**8.2062E+02**	**2.1711E+01**
F5	5.0418E+02	7.2160E−01	5.0425E+02	**5.1916E−01**	5.0586E+02	1.1990E+00	5.0422E+02	6.1627E−01	**5.0410E+02**	8.1315E−01	8.0620E+03	1.9720E+02
F6	6.0096E+02	1.8792E−01	6.0410E+02	2.2901E−01	6.0755E+02	6.6978E−01	6.0090E+02	1.3068E−01	**6.0076E+02**	**9.5338E−02**	3.4838E+07	1.1356E+07
F7	7.0405E+02	2.0114E+00	7.6997E+02	1.2225E+01	8.9401E+02	2.6905E+01	7.0750E+02	4.2654E+00	**7.0074E+02**	**3.3250E−01**	1.1256E+03	7.5035E+01
F8	4.0013E+03	**3.3477E+03**	2.3491E+06	1.4471E+06	1.6032E+08	1.1620E+08	1.7191E+04	3.3261E+04	**2.6354E+03**	4.4907E+03	1.2880E+07	2.8528E+06
F9	9.1358E+02	2.3378E−01	9.1381E+02	2.8076E−01	9.1427E+02	**1.8457E−01**	9.1365E+02	3.1664E−01	**9.1337E+02**	2.2728E−01	1.3412E+03	4.8203E+01
F10	7.5602E+06	**3.1209E+06**	2.9938E+07	1.4513E+07	3.9391E+08	1.6830E+08	1.1337E+07	6.3462E+06	**5.4690E+06**	3.3465E+06	4.3293E+07	2.5289E+07
F11	1.1614E+03	4.1043E+01	1.2889E+03	3.7640E+01	2.1094E+03	4.6591E+02	1.1551E+03	2.8498E+01	**1.1336E+03**	**2.2415E+01**	2.6444E+03	7.8077E+01
F12	2.2417E+03	1.8647E+02	2.9655E+03	2.7960E+02	4.9876E+05	6.1606E+05	2.0617E+03	2.0746E+02	1.7752E+03	1.4548E+02	**1.4048E+03**	**1.7275E+00**
F13	1.7719E+03	2.6527E+01	1.9596E+03	8.1067E+01	3.6329E+03	7.7365E+02	1.7390E+03	2.3388E+01	1.6866E+03	**1.6524E+01**	**1.5734E+03**	8.7998E+01
F14	1.6644E+03	1.8615E+01	1.7522E+03	2.6072E+01	2.1683E+03	1.4580E+02	1.6711E+03	2.8498E+01	**1.6469E+03**	**9.7379E+00**	9.8745E+04	1.1974E+04
F15	2.5522E+03	1.7142E+02	2.8256E+03	**5.5492E+01**	3.9013E+03	4.4677E+02	2.6529E+03	1.6852E+02	**2.4467E+03**	2.1048E+02	3.2739E+04	1.1718E+04

4.3. Large-scale mathematical optimization

In this section, DDAO is compared with Flower pollination algorithm FPA, particle swarm optimization PSO, Firefly algorithm FF, invasive weeds optimization IWO, and Harmony search algorithm on the benchmarks in Table 5 which are Rastrigin's function, the first function of De Jong, Schwefel 2.22 function [85]. The internal parameters for the competitive algorithms are shown in Tables 12–14. As we have mentioned before, DDAO has double function evaluation for each iteration. Thus, the maximum number of function evaluation 1000 was set for the algorithms to optimize the three benchmarks in Table 5 with 51 independent runs except for the results in Table 17 which they were made by 25000 FEs. 1000 FEs for the DDAO algorithm was reached by setting the maximum number of iterations to 100 and the maximum number of sub iterations to 9, i.e. 100*(1+9)=1000. Despite the simplicity of the chosen algorithms in this section, we have chosen them to have extensive comparisons among DDAO and different algorithms with different efficiencies. The statistical results are reported in Tables 15–17. Unlike its competitors, DDAO was able to find the global minimum even for the 50-variable problem variant. It can be seen from the tables that DDAO is considerably more efficient than other algorithms in this experiment. We have obtained the results presented in [86], which are made by 30 independent runs and 25,000 EF on Rastrigin with d=30, to compare the proposed algorithm with these up-to-date results considering the same run conditions. It is clear from Table 18 that DDAO outperforms PSO, BA, FF, MOV, and KH, but it is less powerful than SSA from the perspective of Rastrigin's function.

Table 9
Results of unimodal benchmark functions.

Algorithm		F16	F17	F18	F19	F20	F21	F22
ALO	Best	9316.4579	38.2013	13590.9082	32.8132	3878392.644	9214.17671	1.0462
	STD	3443.7189	1.553E+10	17352.6781	5.6585	8609735.824	3883.6385	2.7229
HHO	Best	1.9206E−246	2.3908E−121	3.7418E−33	2.7836E−119	2.4061E−07	5.1014E−07	4.1574E−06
	STD	0	3.0258E−106	8.9277E−12	1.928E−58	0.0148	8.1745E−05	0.0001
BOA	Best	3.107E−15	4.1484E−31	0.0003	1.1498E−11	28.6863	1.3797	0.0003
	STD	3.0666E−14	1.8788E−08	0.1675	6.2911E−11	0.0401	0.6620	0.0012
DA	Best	44198.9511	66542.3363	59978.0419	4.7826	168609652.6	49583.6652	45.3900
	STD	8018.9132	6.3953E+12	51239.2943	73.8370	36881237.65	6493.0461	30.3442
GWO	Best	1.435E−69	7.7178E−41	0	6.5334E−17	25.6227	5.0682E−05	0.0004
	STD	2.025E−66	1.2384E−38	0	7.9204E−07	0.8873	0.4000	0.0006
PSO	Best	1.775E−08	0.0004	0	0.4981	16.5756	5.1181E−08	0.0390
	STD	9.118E−05	0.0371	32.9629	0.2024	60.2378	6.7098E−05	0.0627
SCA	Best	5.2915E−08	1.0759E−07	1373.2485	3.1864	28.75864405	3.8607	0.0019
	STD	0.27733984	0.0001	9016.2815	9.8578	2216.8213	2.5764	0.0592
WOA	Best	2.32E−206	5.6658E−136	10897.4910	0.0082	25.2287	1.1667E−06	4.4428E−06
	STD	0	2.6327E−122	15429.6748	28.6167	0.4840	6.8412E−06	0.0016
DDAO	Best	1.8314E−05	0.0073	0.0237	0.0286	28.9871	6.3276	0.0095
	STD	34.2240	2.9154	240.7678	0.9151	249.5052	25.7655	0.1160

Table 10
Results of multimodal benchmark functions.

Algorithm		F23	F24	F25	F26	F27	F28
ALO	Best	−5824.4369	213.5864	14.8373	78.54078069	822659.7256	3430510.38
	STD	424.9530	25.7922	0.935792155	36.3289	5814191.034	36456304.3
HHO	Best	−12569.4866	0	8.8817E−16	0	5.2035E−10	1.8301E−09
	STD	549.6215	0	0	0	1.0742E−05	0.000222814
BOA	Best	−847.7902	0	9.9473E−11	0	0.1836	1.9024
	STD	530.4851	0	5.6092E−09	1.75213E−13	0.0867	0.346395574
DA	Best	−3710.2154	359.0001	19.9079	362.5070	198048504.1	580178572.2
	STD	532.8167	28.2561	0.1965	70.64479433	118294722	206364559.1
GWO	Best	−7204.401372	0	7.9936E−15	0	0	0
	STD	837.0003	3.9886	3.0827E−15	0.0034	0.0278	0.2801
PSO	Best	−8622.54301	22.1237	9.13143E−05	5.9955E−09	0	0
	STD	1366.3010	11.9962	0.3275	0.0112	0.0433	0.0114
SCA	Best	−4615.321838	2.786E−05	0.0014	2.1434E−06	0.6462	2.3896
	STD	216.0752	26.6913	9.7933	0.3412	211454.1067	585590.2519
WOA	Best	−12569.4663	0	8.8817E−16	0	0.0045	0.0432
	STD	1041.7665	0	2.5327E−15	0.0035	0.0463	0.2344
DDAO	Best	−4918.7018	1.0672E−06	0.0106	0.0050	0.8890	3.1668
	STD	355.5200	23.9809	1.76871	0.9826	1.9479	199.2664

Table 11
Results of multimodal and fixed-dimension multimodal benchmark functions.

F		ALO	HHO	BOA	GWO	PSO	SCA	WOA	DDAO
F29	Best	0.9980	0.9980	0.9980	0	0	0.9980	0.9980	0.9982
	STD	1.6114	1.1868	0.4505	0	0	0.9565	3.1462	1.7244
F30	Best	0.0004	0.0003	0.0003	0.0003	0.0005	0.0003	0.0003	0.0009
	STD	0.0039	0.0001	5.43E−05	0.0002	0.0001	0.0003	0.0003	0.0015
F31	Best	−1.0316	−1.0316	0.01051	−1.0316	−1.0316	−1.0316	−1.0316	−1.0315
	STD	5.69E−05	1.26E−10	0.3022	7.85E−09	2.37E−16	2.11E−05	4.63E−16	0.0099
F32	Best	0.3978	0.3978	N/A	0.3978	0.3978	0.3979	0.3978	0.3979
	STD	6.5675E−05	3.1178E−06	N/A	0.0001	3.924E−16	0.0012	2.5107E−15	0.0072
F33	Best	3	3	3	3	3	3	3	3.0006
	STD	0.0003	3.7370E−08	0.0009	1.1002E−05	1.4901E−15	3.028E−05	4.4084E−15	0.1296
F34	Best	−3.8627	−3.8627	−2.1872	−3.8627	−3.8627	−3.8622	−3.8627	−3.8621
	STD	0.0001	0.0025	0.3782	0.0027	3.1232E−15	0.0027	0.0025	0.0124
F35	Best	−3.3210	−3.3042	−0.7195	−3.3219	−3.3219	−3.1593	−3.3219	−3.1601
	STD	0.1193	0.0965	0.5051	0.0778	0.0597	0.4687	0.1308	0.0913
F36	Best	−10.0689	−5.0551	−9.5302	−10.1531	−10.1531	−8.1141	−10.1531	−9.0357
	STD	2.7732	0.0073	1.3122	1.7692	3.0631	2.1753	2.2109	1.2049
F37	Best	−10.2757	−10.3460	−8.9743	−10.4029	−10.4029	−8.7401	−10.4029	−6.7537
	STD	3.0858	0.7372	1.0643	1.5937	2.1967	1.8403	2.65421	1.0061
F38	Best	−10.4038	−10.0597	−6.7457	−10.5363	−10.5364	−8.7922	−10.5358	−5.4201
	STD	3.3161	0.6911	0.9682	1.3488	1.7027	1.7019	3.4564	0.8820

Table 12
Setting of DDAO internal parameters.

Parameter	Value
EF	1000
Population size	3
Maximum temperature	50
Cooling rate	0.9995

Table 19 shows the comparison results between DDAO and invasive weeds optimization (IWO) using data given by [87] for 1000 FE. The comparison demonstrates that dynamic differential annealed optimization gives results close to the global minimum for d=30 and d=50. On the other hand, IWO results have a lower standard deviation than our proposed optimization algorithm. We employed another comparison with six optimizers on De Jong 1 function using results obtained from [86], shown in Table 20. The results were done using 30 independent runs with 25,000 FE, and DDAO was run using these conditions for a fair comparison. DDAO has better performance than the first five algorithms in Table 9 but worse performance than SSA; this is the same as for the previous comparison on Rastrigin's function.

De Jong 1 also was optimized by invasive weeds optimization algorithm in [87], and the results for d=30 and d=50, and the results are used for comparison with DDAO as shown in Table 21. IWO has a reproduction section that distributes a random number of seeds over random positions in the search space using standard deviation formulation to generate new offspring [88]. DDAO uses sub-iteration, which is a simple random search within the search area, to create a random solution which is later operated with other existing ones in the population. These differing random search mechanisms are the reason behind the results of Table 21, where DDAO has better convergence to the global than IWO and a worse Std value F40 function.

Another comparison with d=30 and FE= 25,000 is made among DDAO and other algorithms shown in Table 22 on Schwefel 2.22, where other algorithms' results are taken from [86]. Also, as we have seen for Rastrigin and De Jong 1, DDAO has best results than the other algorithms in Table 22 with the exception of SSA.

In most of the cases, the standard deviation of DDAO is higher than that of other metaheuristics, and this is expected because of the random search of the proposed algorithm.

4.4. Small-scale mathematical optimization

DDAO was tested on seven benchmark functions shown in Table 6 [84], namely Bohachevsky, Booth, Matyas, Easom, Beale, Schaffer N.2, and Six-hump Camel, where all have d=2. During this study, we have compared the proposed algorithm using test results available in [86], where the test has been done using 30 independent runs with 25,000 FE on these test functions. The test was carried out with PSO, BA, FF, MOV, KH, and SSA, and we have employed the same run conditions to compare DDAO results. Also, we have found source code for SMO online at http://smo.scrs.in/ and added its results to the comparative study in Table 23 using the same run conditions in this section. For small dimension problems, the good performance of DDAO is proved by the statistical results listed in Table 23.

Table 14
Internal parameters of IWO.

Parameter	Value
Initial population size	25
FE	1000
Maximum size of plant population	25
Maximum number of seeds	5
Minimum number of seeds	0
Nonlinear modulation index	3
Initial value of standard deviation	0.5
Final value of standard deviation	0.001

Table 15
Comparison of statistical optimization results for F39 test function.

Algorithm	Best	Worst	Std	Mean
DDAO	**0.14395**	**259.23331**	63.39936	**46.78938**
FPA	476.4448214	695.2391047	42.48481832	600.1606581
PSO	332.0644542	505.2325899	42.34850018	403.2702636
FF	320.79241	512.83243	43.31045	404.13040
HS	211.16856	301.53563	**21.97571**	250.45531

Table 16
Comparison of statistical optimization results for the F40 function.

Algorithm	Best	Worst	Std	mean
DDAO	**3.3512E−09**	**2.37044**	**0.37354**	**0.16872**
FPA	82.24810	181.62935	25.04960	139.25594
PSO	22.28837	62.82966	8.22995	40.39927
FF	69.97707348	198.9227655	25.87562647	129.1688462
HS	36.32790	70.14757	8.04909	52.41646

5. Path planning application

DDAO has also been applied to practical engineering problems such as path planning [89]. Path planning is a fundamental optimization problem in robotics, and it has received a great deal of consideration over the last few decades. The best usage of path planning can be seen in industrial robots [90] for the automotive industry and particularly self-driving vehicles [91]. Path planning is the problem of finding the shortest path between two points in a free-of-collision environment or within a collision environment. The shortest path between two points on a curve in a free-of-obstacle workspace is just a straight line, while the problem gets complicated when the workspace contains obstacles. Consider the workspace shown in Fig. 6; there is a constrained environment with coloured circular obstacles. The workspace is limited such that $x \in [-10, 30]$ and $y \in [-10, 30]$mm, and the constraints are as shown in Table 24. In this example, a cubic spline curve was used to generate the desired path with a smooth nature. To

Table 13
Internal parameters of FA, FPA, HS and PSO used for the test functions.

	Parameter	Value		Parameter	Value
FA	EF	1000	FPA	EF	1000
	No. of population	25		No. of population	25
	Light Absorption Coefficient	1		Probability switch	0.8
	Parameter	Value		Parameter	Value
	EF	1000		EF	1000
	Harmony memory size	25		No. of population	25
HS	Harmony consideration rate	0.9	PSO	Personal acceleration	1
	Minimum pitch adjusting rate	0.4		Social acceleration	1
	Maximum pitch adjusting rate	0.9		Inertia coefficient	1
				Damping	0.95

Table 17
Comparison of statistical optimization results for the F41 test function.

Algorithm	Best	Worst	Std	Mean
DDAO	**0.0942726**	**22.549416**	**5.488026**	**5.843656**
IWO	143350.5749	3.1262E+15	4.746E+14	1.00081E+14
FPA	1640.5115	8.30132E+15	1.25043E+15	2.80783E+14
PSO	52.39179	161.60235	24.85333	104.4510
FF	69.05435	710328407.4	99463169.54	14474339.4
HS	37.04751	69.69666	7.12440	55.42751

Table 18
Detail of statistical performance of SSA and other metaheuristics [86] and DDAO on F39 test function.

	PSO	BA	FF	MOV	KH	SSA	DDAO
Best	6.0234E+01	4.6766E+01	1.7612E+01	6.7986E+01	3.0381E+00	0	1.19155E−07
Worst	1.5111E+02	2.3581E+02	4.9853E+01	2.0129E+02	2.4926E+01	7.6657E−06	0.711162199
Mean	1.0309E+02	1.2192E+02	2.5069E+01	1.1867E+02	1.2391E+01	4.9059E−07	0.180299062
STD	2.4727E+01	3.9334E+01	6.9514E+00	3.3984E+01	5.4147E+00	1.5057E−06	0.097915887

Table 19
Details of the statistical performance of IWO [87] and DDAO on F39 test function.

Algorithm	d	Best	Std
DDAO	30	**1.47895E−04**	45.0609
	50	**0.14395**	63.39936
IWO [87]	30	6.697E+01	**1.655E+01**
	50	1.590E+02	**3.261E+01**

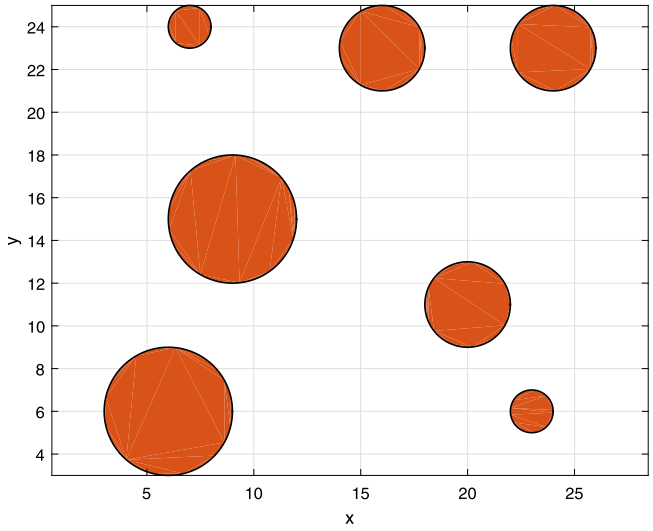

Fig. 6. The constrained workspace of the path planning problem.

find the optimal path by an optimization algorithm the following procedure applies:

1. Define three variables to represent the control points on the cubic spline. Each variable is a structure of two sub-variables representing the *x*-axis and *y*-axis for the control point, so in total, we have six variables, i.e. three x-axes and three y-axes for the three points.

2. Set random variables for the six optimization variables according to side constraints.

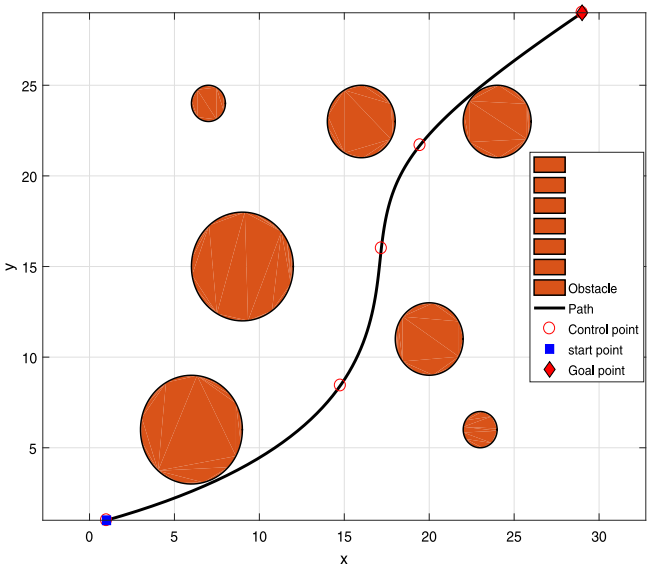

Fig. 7. Proposed path connecting start and goal points using DDAO.

3. A cubic spline is fitted onto the control points as well as start and goal points, using the specified (x,y) coordinates. Basically, the construction of the cubic spline curve takes the form

$$s_i = a_i (x - x_i)^3 + b_i (x - x_i)^2 + c_i (x - x_i) + d_i. \quad (6)$$

By fixing the coefficients a_i, b_i, c_i, and d_i, the entire curve is fixed by using boundary conditions for Not-a-knot spline where the five points, start, goal, and three control points represent these boundary conditions. In this study, we have used MATLAB Toolbox to find the curve's given points.

4. Use the obtained cubic spline curve to generate (for example) 100 equally spaced points along the curve. MATLAB uses Not-a-knot spline [92] represented by the function *spline( )* [93] which is used in code implementation for this study. It is possible to use a 4th polynomial instead of the cubic spline curve, but we have used the cubic spline curve in this study for simplicity because MATLAB calculates it automatically.

Table 20
Details of the statistical performance of SSA and other metaheuristics [86] and DDAO on F40.

	PSO	BA	FF	MOV	KH	SSA	DDAO
Best	2.4402E+02	1.9341E+04	4.7333E−03	4.0172E−01	1.0942E−02	**7.9225E−20**	6.73016E−07
Worst	2.7319E+03	6.6267E+04	2.4806E−02	1.5333E+00	2.1477E−01	**5.7411E−07**	0.315399914
Mean	1.3576E+03	3.9384E+04	1.1597E−02	7.8582E−01	5.7558E−02	**4.1689E−08**	0.086905074
STD	6.4254E+02	1.0736E+04	4.3201E−03	2.4795E−01	5.0330E−02	**1.4356E−07**	0.087327069

Table 21
Detail of statistical performance of IWO [87] and DDAO on F40 test function.

Algorithm	d	Best	Std
DDAO	30	**7.72938E−11**	53.0884
	50	**3.3512E−09**	0.37354
IWO [87]	30	7.554E−04	**8.575E−05**
	50	2.736E−03	**3.284E−04**

5. Evaluate the distance between each successive point on the curve and sum the distances to calculate the total length of the curve. The following is the objective function of the path planning problem that should be minimized:

$$O = \sum_{i=1}^{n} \sqrt{(x_{i-1} - x_i)^2 + (y_{i-1} - y_i)^2} + penalty \tag{7}$$

where O is the objective (length of the curve), $i = 1, \ldots, n$. The static penalty was used for path planning problem, where it should have high value when the path contacts any obstacle in the workspace; otherwise, it has zero value.

6. If the spline curve collides with an obstacle, set the static penalty to a high value, for example, 1000, if not set it to zero.

7. Repeat the process each time the optimizer updates the solution and stop the optimization when the limit number of iterations is reached.

The solution for the problem in Fig. 6 is represented in Fig. 7 which represents the shortest path between point (1,1) and point (29,29). Table 25 shows the competition among DDAO, PSO, FPA, FF, HS, and IWO to find the minimum distance between start and goal points explained in Fig. 6. For this study, ten independent runs were used with 10,000 FE. DDAO found a shorter path than FPA, HS, and IWO did, while FF and PSO returned the best results amongst all algorithms.

Table 24
Definition of circular obstacles in the two-dimensional path programming problem.

No.	x-axis	y-axis	Radius
1	6	6	3
2	20	11	2
3	23	6	1
4	9	15	3
5	7	24	1
6	16	23	2
7	24	23	2

Table 25
Comparison of DDAO with PSO, FPA, FF, HS, and IWO on the path planning problem.

	PSO	FPA	FF	HS	IWO	DDAO
Best	**40.6636**	43.2796	40.8410	43.0543	43.2263	42.6742
Worst	43.5168	55.9330	**43.2421**	44.0771	53.7578	47.4207
Mean	42.2716	46.4287	**42.0463**	43.6001	47.2505	45.8608
STD	1.1275	4.1464	1.2253	**0.3036**	2.9872	1.4902

Table 22
Comparison of DDAO with SSA and other metaheuristics on F41, data obtained from [86].

	PSO	BA	FF	MOV	KH	SSA	DDAO
Best	1.0967E+01	6.9978E+01	1.9719E−01	3.1692E−01	2.2580E+01	2.2266E−08	0.008369553
Worst	2.4748E+01	1.6143E+07	7.1570E−01	6.0719E+01	9.0126E+01	7.1423E−03	0.50157453
Mean	1.8341E+01	6.3387E+05	3.7229E−01	4.2299E+00	4.5339E+01	5.1849E−04	0.135183166
STD	3.2867E+00	2.9400E+06	1.1885E−01	1.3708E+01	1.5991E+01	1.4144E−03	0.155689847

Table 23
Detail of statistical performance of SSA and other metaheuristics [86] and DDAO in small-scale two-variable test functions.

Function		PSO	BA	FF	MOV	KH	SSA	SMO	DDAO
F42	Best	4.4298E−14	1.6438E+00	4.9443E−08	1.0021E−05	2.8890E−08	0	2.1330E−10	9.64105E−07
	Worst	1.3643E−09	4.3439E+02	9.1103E−06	1.5693E−03	1.4369E−06	0	9.8995E−09	0.103384664
	Mean	1.4898E−10	6.9719E+01	3.4682E−06	3.5952E−04	2.2989E−07	0	4.2405E−09	0.030674913
	STD	2.7559E−10	1.1194E+02	2.6636E−06	3.7081E−04	2.9457E−07	0	3.1099E−09	0.018695917
F43	Best	1.3482E−14	3.0619E−11	2.7382E−10	4.2336E−09	5.9289E−12	1.2622E−29	2.7652E−10	4.4529E−05
	Worst	1.2419E−09	2.0412E−09	1.9998E−08	2.9865E−06	1.5568E−09	8.1255E−24	1.0126E−08	0.07502273
	Mean	8.7865E−11	6.7996E−10	5.4966E−09	5.8026E−07	1.9913E−10	9.5859E−25	4.4779E−09	0.01767008
	STD	2.3629E−10	5.4689E−10	4.8029E−09	6.5125E−07	3.1189E−10	1.7996E−24	2.9068E−09	0.01952182
F44	Best	8.6209E−17	1.4036E−12	2.9676E−12	2.0125E−10	2.1689E−14	1.5111E−29	3.5402E−12	5.66862E−12
	Worst	7.1849E−12	2.2319E−10	1.2805E−09	4.0157E−08	6.8392E−11	2.0707E−24	9.7150E−09	1.11537E−05
	Mean	8.4738E−13	2.9659E−11	3.1396E−10	1.3148E−08	8.8694E−12	1.542E−25	3.1465E−09	2.2285E−06
	STD	1.5676E−12	4.8089E−11	2.9301E−10	1.1549E−08	1.2854E−11	4.7571E−25	3.1259E−09	9.8777E−07
F45	Best	−1	−1	−1	−1	−1	−1	−1	−0.99843
	Worst	−1	0	0	0	0	−1	−0.9999	−0.40402
	Mean	−1	−3.3347E−02	−7.3333E−01	−9.6664E−01	−9.6666E−01	−1	−1	0.180368
	STD	2.8316E−11	1.8257E−01	4.4978E−01	1.8257E−01	1.8257E−01	0	3.0893E−09	−0.80388
F46	Best	1.3624E−14	2.8649E−11	1.2360E−11	9.2179E−09	4.4607E−13	2.1832E−29	2.8207E−11	9.06E−05
	Worst	4.9376 E-01	7.6207E−01	3.9985E−09	7.6207E−01	1.5103E−09	2.7633E−20	8.9618E−09	0.012256
	Mean	6.2888E−02	1.5241E−01	8.4259E−10	5.0805E−02	1.7322E−10	9.5584E−22	2.9834E−09	0.002924
	STD	1.6315E−01	3.1004E−01	7.7095E−10	1.9334E−01	3.0481E−10	5.0400E−21	2.7488E−09	0.003105
F47	Best	0	9.7159E−03	4.6615E−03	2.1517E−06	1.2317E−07	0	1.9978E−10	1.7525E−09
	Worst	9.7159E−03	3.7329E−01	3.8682E−02	9.7159E−03	2.0569E−05	9.7159E−03	9.7903E−09	1.1726E−05
	Mean	3.5625E−03	1.4118E−01	1.1809E−02	1.3113E−03	3.3604E−06	9.7159E−04	3.9342E−09	2.77E−06
	STD	4.7621E−03	1.0699E−01	7.3682E−03	3.3529E−03	4.1334E−06	2.9646E−03	3.0094E−09	1.6223E−06
F48	Best	−1.03163	−1.03163	−1.03163	−1.03163	−1.03163	−1.03163	−1.0316	−1.03157
	Worst	−1.03163	−0.21546	−1.03163	−1.03163	−1.03163	−1.03163	−1.0316	−1.02894
	Mean	−1.03163	−0.95001	−1.03163	−1.03163	−1.03163	−1.03163	−1.0316	0.000651
	STD	8.2049E−11	2.4904E−01	3.9448E−09	1.6383E−07	6.3582E−10	4.5168E−16	3.2103E−09	−1.03066

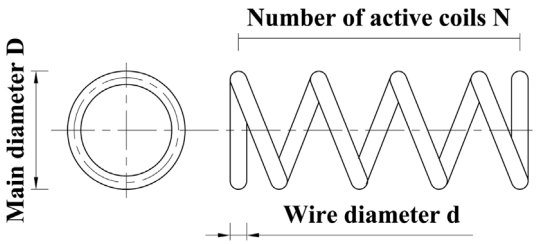

Fig. 8. Closed coil helical spring design parameters.

6. Spring design problem

In this section, the performance of DDAO was examined the spring design problem. This problem is constrained optimization and has been conducted by many researchers in the literature, so this is a further opportunity to compare the proposed algorithm with other state-of-the-art optimizers. The formulation of the constrained objective functions of the spring design problem was considered as in [18], and the source code can be found in [94]. The experiment was performed over 30 independent runs and 500 iterations; these parameters were chosen to be consistent and fair with other state-of-the-art optimization algorithms. In this section, only DDAO has been run while all other data of comparison, which are shown in Table 26, have been taken from the literature. The main variables for this engineering problem are illustrated in Fig. 8; diameter of the wire d, coil diameter D, and number of active coils N. Chaotic gray wolf optimization (CGWO) [18] has been compared with GWO [17], CSA [95], GA3 [96], GA4 [97], CPSO [98],HPSO [99], G-QPSO [100], PSO [100], DSS-MDE [101], PSO-DE [102], SC [103], UPSO [104], $(\mu+\lambda)-ES$ [105], ABC [106], TLBO [107], and MBA [108]. The comparison data on this problem have been taken from [18] and compared with DDAO results. The problem can be declared as:

Consider

$$\text{Consider } \vec{x} = \begin{bmatrix} x_1 & x_2 & x_3 \end{bmatrix} = \begin{bmatrix} d & D & N \end{bmatrix},$$

$$\text{minimize } f\left(\vec{x}\right) = (x_3+2)\,x_2x_1^2, \tag{8}$$

$$\text{subject to } \quad g_1\left(\vec{x}\right) = 1 - \frac{x_2^3x_3}{717854x_1^4} \le 0,$$

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

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

$$g_4\left(\vec{x}\right) = \frac{x_1+x_2}{1.5} - 1 \le 0,$$

$$\text{variable range} \quad 0.05 \le x_1 \le 2.00,$$
$$0.25 \le x_2 \le 1.3,$$
$$2.00 \le x_3 \le 15.00.$$

By observing the comparison results in Table 26, we can notice that CGWO has the minimum cost value; however, these results need further clarification. Table 27 shows examination for the solution vector of DDAO and CGWO; the values of the constraints should be all less than or equal zero, as mentioned in Eq. (8). CGWO has violated the constraint$g_2\left(\vec{x}\right)$, so the minimum cost function of this algorithm could not be the optimum value. The solution vector of DDAO is feasible since it did not violate the conditions of the constraints.

7. Sensitivity to the internal parameters of DDAO

In this section, the focus is on the effect of the internal parameter of DDAO algorithm on the convergence sensitivity. The steady relationship between optimality and the number of iterations is the same as with other optimization algorithms, where increasing the number of iterations should lead to best results, while also being more time-consuming. For a fixed number of iterations equal to 500, we studied the effect of sub-iterations, initial temperature, cooling rate, and the population size on the convergence over Schwefel 2.22 test function with d=30 and 51 independent runs. For study forging parameter, run conditions are expressed in 7.4.

Table 26
Comparing DDAO with CGWO and other algorithms from [18].

Algorithm	Worst	Mean	Best	Std
DDAO	0.0173199	0.0151829	0.0129065	0.00126
CGWO	0.0121791	0.0121749	0.0119598	1.039E−05
GWO	0.0122515	0.0121836	0.0126660	1.085E−05
CSA	0.0126701	0.0127690	0.0126650	1.357E−06
GA3	0.0128220	0.0127690	0.0127048	3.940E−05
GA4	0.0129730	0.0127420	0.0126810	5.90E−05
CPSO	0.0129240	0.0127330	0.0126747	5.20E−04
HPSO	0.0127190	0.0127072	0.0126652	1.58E−05
G-QPSO	0.0177590	0.0135240	0.0126650	0.001268
QPSO	0.0181270	0.0138540	0.0126690	0.001341
PSO	0.0718020	0.0195550	0.0128570	0.011662
DSS-MDE	0.0127382	0.0126693	0.0126652	1.25E−05
PSO-DE	0.0126653	0.0126652	0.0126652	1.2E−08
SC	0.0167172	0.0129226	0.0126692	5.9E−04
UPSO	N.A.	0.0229400	0.0131200	7.2E−03
$(\mu+\lambda)-ES$	N.A.	0.0131650	0.0126890	3.9E−04
ABC	N.A.	0.0127090	0.0126650	0.01281
TLBO	N.A.	0.0126657	0.0126650	N.A.
MBA	0.0129000	0.0127130	0.0126650	6.3E−05

N.A. – Not Available.

7.1. Effect of sub-iterations

The study was carried out using fixed values of number iterations equal to 500, initial temperature 50, cooling rate 0.9995, and population size 25. The sub-iteration parameter represents the random search behaviour of the proposed algorithm, where the best solution among all the sub-iterations will be used in Eq. (3) to produce a candidate solution. Since the process is random, it is hard to predict precisely what will happen in the next sub-iteration, and that is the reason behind the relatively high value of the standard deviation of DDAO solutions. The algorithm is designed to go through the global minimum of a function that is desired for many engineering applications, especially structural and mechanical design. On this basis, increasing the number of sub-iterations will increase the chances to find the nearest values to the global minimum, and that is the reason why DDAO is more likely than other algorithms to find the minimum of a function. Table 28 gives the effect of different numbers of sub-iterations on convergence behaviour.

7.2. Effect of initial temperature and cooling rate

Cooling rate and initial temperature are the elements of the simulated annealing algorithm, as explained in Section3. Starting with high temperature ensures that more candidate solutions can be accepted. A slow cooling rate ensures that the algorithm reaches the nearest point to global while a high cooling rate makes the algorithm more likely to skip the global or near-global solution, according to previous studies illustrating the simulated annealing algorithm since it was published in 1984 [41,49].

Table 27
Violations of the best solution vectors of FO and CGWO algorithms to the constraints.

Algorithm	$(\vec{x}_*)^T$	$g_1(\vec{x})$	$g_2(\vec{x})$	$g_3(\vec{x})$	$g_4(\vec{x})$	No. of violations
DDAO	0.054638 0.42988 8.057	−0.00048296	−0.0038741	−4.154	−0.67699	0
CGWO	0.052796 0.804380 2.000000	−0.8663	0.9018	−4.7302	−0.4285	1

Table 28
Effect of the sub-iteration parameter of DDAO.

	Sub iterations					
	2	5	10	15	20	25
Best	0.033744555	0.035906041	0.0117023	0.004161772	0.004938077	0.003149992
Worst	3.181287015	2.381758846	1.772891239	1.391565474	1.563270793	2.898306845
Mean	0.635670365	0.654853547	0.457358637	0.365519647	0.383818532	0.668572373
STD	0.613063483	0.641435961	0.492672051	0.420029606	0.417637429	0.585375833

7.3. Effect of population size

The random behaviour of DDAO makes it independent of population size, as can be seen from Table 29. The experiment was done using iteration number 500, 10 sub-iterations, the initial temperature of 1000, cooling rate 0.99, and different population sizes. Returning to Eq. (3), which is the essential part of the search engine, the best solution of sub-iteration is mathematically operated with at least two solutions from the population. It is clear from Table 29 that DDAO is not strongly affected by whether the population size is 2 or 100.

7.4. Effect of forging parameter f

At the end of Section 3, we have presented a justification to invent the forging parameter, and in this section, we will discuss this new parameter numerically. DDAO was run 51 times with a maximum number of iterations 500, sub iterations 300, and 3 population size on five benchmarks from Tables 6 and 7, the results are reported in Table 30 where *rand* means random number between 0 and 1. Each run was repeated twice: one run with f= random number and one run with $f = 1$, and we have done that to study the effect of using or not using f on Eq. (3). A bunch of benchmarks have well-performed by DDAO with $f = 1$ and poorly performed when f= random number; the vis versa is correct for another bunch of functions. Thus, forging technique may represent an average value between using or not using f on Eq. (3). As shown in Table 30, we can notice that problem on five benchmarks when DDAO has better performance with f=rand instead of $f = 1$ on F47 and F49. For the other hand, forging technique can weaken the DDAO on some set of problems which is already good on them. Still, we have accepted using it as an overall improvement on the comprehensive set of benchmarks and applications. However, we believe that this issue should have great attention and analysis for future work.

8. Exploration and exploitation analysis

Recalling paragraph 6 in Section 3. The annealed process alone of DDAO, which is represented by the acceptance criterion in Eqs. (4) and (5), gives a high rate of exploration and low rate of exploitation at the high value of temperature. The exploration rate decreases, and the exploitation rate increases while the temperature drops down. A solution can be accepted if it introduces an improvement to the existing solutions in the population, or it can be accepted if the acceptance criteria in Eq. (4) is greater than a random number $\in [0, 1]$. At the beginning of the search, the temperature starts with high value; consequently, the acceptance criteria (P) will be close to one, according to Eq. (4). This means that a wide range of random numbers can be less that one, and the solution will be selected. At a low temperature, the acceptance criteria will be close to zero; according to Eq. (4), this means that a very narrow range of random numbers could be less than P and the solution is less likely to be selected, for instance, $e^{-1/300} = 0.9967$while $e^{-1/0.3} = 0.0357$. We have used F33, F29, and F22 as examples to reveal the convergence of the DDAO algorithm as shown in Fig. 9. The variable size is set to two for the three benchmarks, and regardless the simplicity, we would like to focus on the convergence behaviour to prove the above-mentioned information in this section. In Figs. 10–12 part (c), we can see how the number of accepted solutions start with high numbers and reduce over iterations due to applying equation (4). Despite the rapid convergence at the beginning of run time, as shown in Figs. 10–12 part (a), the algorithm still provide solutions around the global optimum as revealed in parts (b) of the figures. This happens due to the random search using sub iterations stage and unbalanced rates of exploitation and exploration at low temperature. In other words, we can say that the rate of exploitation is insufficient to drive the algorithm to the exact position of the global optimum. Also, in Fig. 11(b), one can see the DDAO can escape efficiently from local optima but still fluctuate about the global. This is the same case for large scale problems and large search spaces with a drawback that DDAO needs more iterations and time to reach a nearby position to the global point in the search space. This behaviour is responsible for the relatively high value of standard deviation for the solutions of the DDAO. The final user, especially in fields of engineering applications, which may be not caring too much about standard deviation, should expect time-consuming for large and complicated engineering problems to get better solutions. These disadvantages should be considered while using DDAO, and it can be modified in future works to fix these drawbacks.

9. The time complexity of DDAO

The first function of De Jong is used in this section to study the effect of increasing the size of a problem on the algorithm performance in terms of time. Big O notation is used to describe the performance of the DDAO while optimizing F40 with DDAO parameters: maximum number of iterations =1, maximum number of sub iterations =100, population size=3, initial temperature 2000, cooling rate 0.95, and five values for problem size dim=[10,100,1000,10000,100000].

The question: How much time does it takes to run optimization algorithm and return cost value? Is depending on many

Table 29
Effect of the sub-iteration parameter of DDAO.

	Population size					
	2	5	10	20	50	100
Best	0.004588807	0.003294644	0.002975413	0.001128964	0.003799432	0.005931364
Worst	5.782867964	2.860368544	2.99193603	2.18412072	2.537364465	3.321913183
Mean	1.061361124	0.646818514	0.60061176	0.512118794	0.509958543	0.494545148
STD	1.022380725	0.689944748	0.612057671	0.554569434	0.476632874	0.479707522

Table 30
Effect of forging parameter of DDAO.

Function	*f*	Best	Worst	Mean	STD
F49	*rand*	**−0.999999**	**−0.995957**	**−0.999747**	**0.000686**
	1	−0.999866	−0.948807	−0.991981	0.009472
F50	*rand*	**−959.6363496**	−934.0686608	−953.2524047	4.975861686
	1	−959.6222867	**−949.6176215**	**−955.9135739**	**2.410915576**
F51	*rand*	−19.2084528	−19.1893493	−19.2043446	0.003935819
	1	**−19.208483**	**−19.2025163**	**−19.206841**	**0.001390003**
F52	*rand*	**5.7975E−05**	0.074235653	0.016777198	0.018005424
	1	8.86123E−05	**0.048659163**	**0.008870316**	**0.0102388**
F47	*rand*	**3.17633E−10**	**4.68435E−05**	**3.09842E−06**	**7.64219E−06**
	1	9.08394E−08	0.00689151	0.001671139	0.001899747

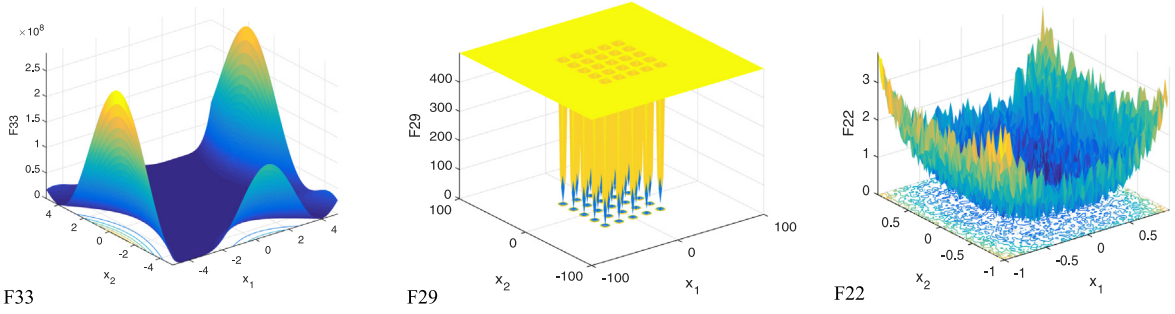

Fig. 9. Topology of F33, F29, and F22 benchmarks.

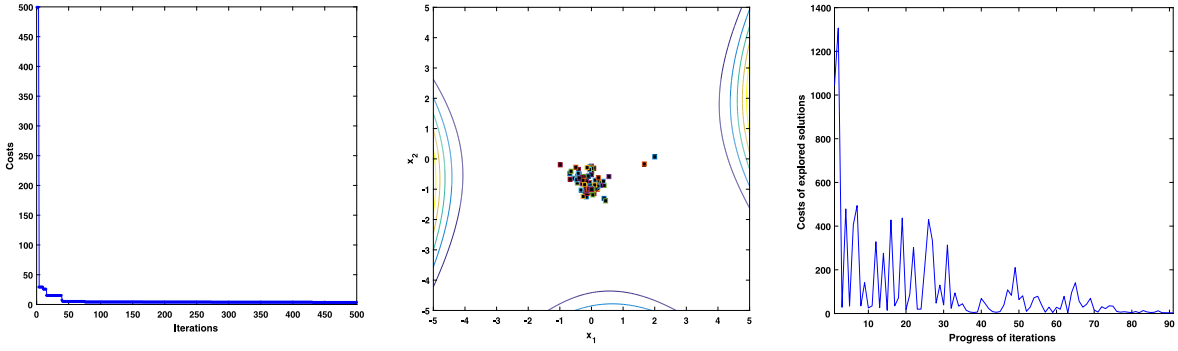

Fig. 10. Convergence on F33: (a) convergence curve, (b) convergence history, and (c) explored solutions vs iterations.

factors like processor type of the computer, the algorithm runs in parallel with other programs or not, virus scanners, the effect of flushing buffers, throttling of the processor, and which programming language is used [109]. For example, we have done the experiment using MATLAB R2016b where it is required to find the elapsed time to get the summation (sum)of a vector (x) of length 1000 with random elements of values between −10 and 10, and we have measured the time using tic toc as shown in the following code:

```
clear
clc;
d=1000;
varmin = -10;
vVarmax = 10;
x=unifrnd(varmin,vVarmax,[1 d]);
tic
sum(x);
toc
tic
sum(x);
toc
tic
sum(x);
toc
```

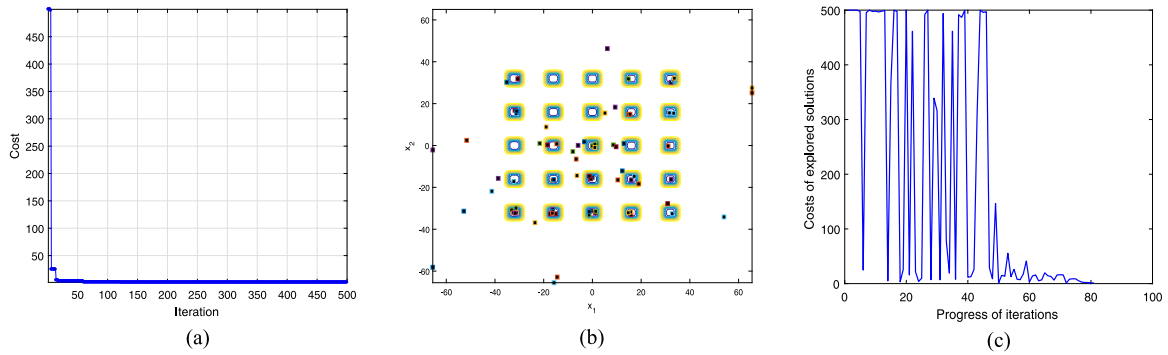

Fig. 11. Convergence on F29: (a) convergence curve, (b) convergence history, and (c) explored solutions vs iterations.

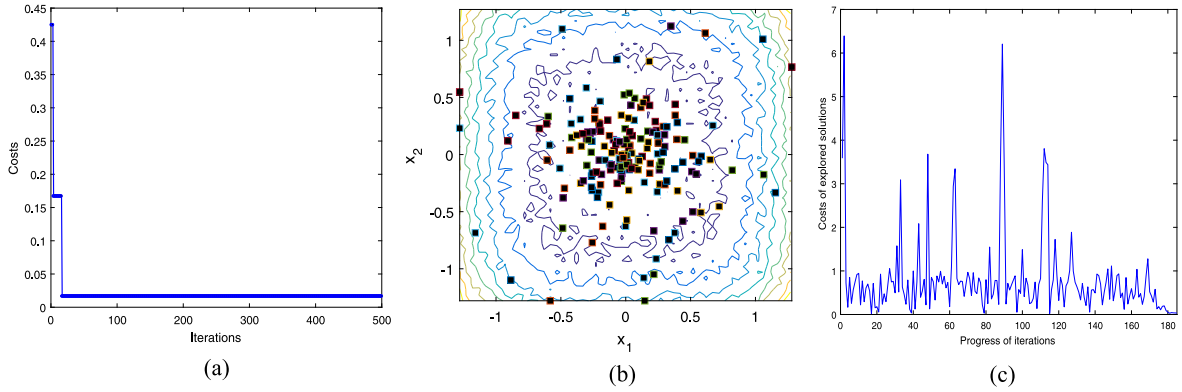

Fig. 12. Convergence on F22: (a) convergence curve, (b) convergence history, and (c) explored solutions vs iterations.

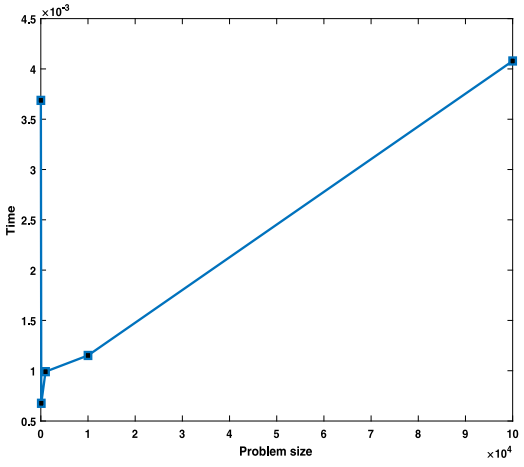

Fig. 13. Runtime vs problem size for sub iterations=1.

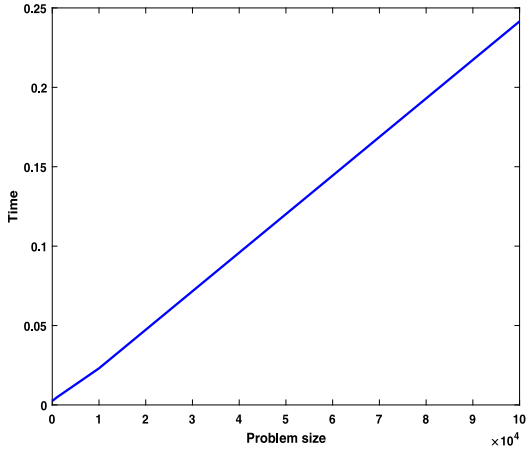

Fig. 14. Runtime vs problem dimension for sub iterations=100.

```
tic
sum(x);
toc
```

The results are:

Elapsed time is 0.000105 s.
Elapsed time is 0.000052 s.
Elapsed time is 0.000048 s.
Elapsed time is 0.000047 s.

The random nature of the results is due to the effect of flushing buffers, and one solution to reduce this random effect is by using a loop, and that is why we have set a maximum number of sub iterations to 100. For more clarification, Fig. 13 shows meaningless linear relation between problem size and runtime for sub iterations =1 with random effect at small problem sizes. For the other hand, Fig. 14 shows a rational relation between runtime and problem size for subiterations=100.

In the same meaning, the procedure of calculating the complexity of an algorithm in CEC2015 special session [80] uses a loop of length 1000000. However, for the facts revealed in this section, setting numerical value for the complexity of an algorithm is improper way. Thus, the best question could be: How does the runtime of an algorithm grow? and this question can be answered by using big O notation [110]. By using this notation, the complexity of an algorithm can be described regardless of the type of the computer or any hardware restrictions. As illustrated in Fig. 14, we can see that the time complexity of the DDAO is a linear time $O(n)$ where n is the problem size.

10. Conclusion

This study presented a novel metaheuristic optimization algorithm; dynamic differential annealed optimization (DDAO), based on simulated annealing and inspired by the production process of dual-phase (DP) high strength steel. In producing DP steel, there is an overall differential cooling accompanied by mechanical deformation and this behaviour has been mathematically modelled to propose the new optimization algorithm DDAO. The new algorithm was tested using 51 test functions and compared to a large number of optimization algorithms. Remarkably, the present algorithm outperformed the some of the metaheuristics in many cases and mostly converged to the target global optimum or a very close solution. Furthermore, DDAO also efficiently solved a constrained path planning problem and spring design problem. It is worth mentioning that the differential technique highly improves the performance of the algorithm, a strategy has been used previously to produce the differential evolution algorithm; however, our algorithm employs it in a different way.

CRediT authorship contribution statement

Hazim Nasir Ghafil: Conceptualization, Methodology, Software, Validation, Formal analysis, Investigation, Resources, Data curation, Writing - original draft, Writing - review & editing, Visualization. **Károly Jármai:** Supervision, Funding acquisition, Project administration.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Appendix A. Supplementary data

Supplementary material related to this article can be found online at https://doi.org/10.1016/j.asoc.2020.106392.

References

[1] X.-S. Yang, Nature-Inspired Algorithms and Applied Optimization, vol. 744, Springer, London, 2017, http://dx.doi.org/10.1007/978-3-319-67669-2.

[2] S. Li, H. Chen, M. Wang, A.A. Heidari, S. Mirjalili, Slime mould algorithm: A new method for stochastic optimization, Future Gener. Comput. Syst. (2020).

[3] S. Kirkpatrick, C.D. Gelatt, M.P. Vecchi, Optimization by simulated annealing, Science 220 (4598) (1983) 671–680.

[4] A. Kaveh, A. Dadras, A novel meta-heuristic optimization algorithm: thermal exchange optimization, Adv. Eng. Softw. 110 (2017) 69–84.

[5] Y.-H. Kim, Y. Yoon, Z.W. Geem, A comparison study of harmony search and genetic algorithm for the max-cut problem, Swarm Evol. Comput. 44 (2019) 130–135.

[6] E. Rashedi, H. Nezamabadi-Pour, S. Saryazdi, GSA: a gravitational search algorithm, Inf. Sci. 179 (13) (2009) 2232–2248.

[7] S. Mirjalili, S.M. Mirjalili, A. Hatamlou, Multi-verse optimizer: a nature-inspired algorithm for global optimization, Neural Comput. Appl. 27 (2) (2016) 495–513.

[8] S. Mirjalili, SCA: a sine cosine algorithm for solving optimization problems, Knowl.-Based Syst. 96 (2016) 120–133.

[9] A. Faramarzi, M. Heidarinejad, B. Stephens, S. Mirjalili, Equilibrium optimizer: A novel optimization algorithm, Knowl.-Based Syst. 191 (2020) 105190.

[10] A. Tabari, A. Ahmad, A new optimization method: electro-search algorithm, Comput. Chem. Eng. 103 (2017) 1–11.

[11] M.D. Li, H. Zhao, X.W. Weng, T. Han, A novel nature-inspired algorithm for optimization: Virus colony search, Adv. Eng. Softw. 92 (2016) 65–88.

[12] Z.A.A. Alyasseri, A.T. Khader, M.A. Al-Betar, M.A. Awadallah, X.-S. Yang, Variants of the flower pollination algorithm: a review, in: Nature-Inspired Algorithms and Applied Optimization, Springer, 2018, pp. 91–118.

[13] N.A. Kallioras, N.D. Lagaros, D.N. Avtzis, Pity beetle algorithm–a new metaheuristic inspired by the behavior of bark beetles, Adv. Eng. Softw. 121 (2018) 147–166.

[14] C.H. Silva-Santos, P. Goulart, F. Bertelli, A. Garcia, N. Cheung, An artificial immune system algorithm applied to the solution of an inverse problem in unsteady inward solidification, Adv. Eng. Softw. 121 (2018) 178–187.

[15] S. Mirjalili, The ant lion optimizer, Adv. Eng. Softw. 83 (2015) 80–98.

[16] S. Mirjalili, P. Jangir, S. Saremi, Multi-objective ant lion optimizer: a multi-objective optimization algorithm for solving engineering problems, Appl. Intell. 46 (1) (2017) 79–95.

[17] S. Mirjalili, S.M. Mirjalili, A. Lewis, Grey wolf optimizer, Advances Eng. Softw. 69 (2014) 46–61.

[18] M. Kohli, S. Arora, Chaotic grey wolf optimization algorithm for constrained optimization problems, J. Comput. Des. Eng. 5 (4) (2018) 458–472.

[19] S. Mirjalili, S. Saremi, S.M. Mirjalili, LdS Coelho, Multi-objective grey wolf optimizer: a novel algorithm for multi-criterion optimization, Expert Syst. Appl. 47 (2016) 106–119.

[20] S. Mirjalili, Dragonfly algorithm: a new meta-heuristic optimization technique for solving single-objective, discrete, and multi-objective problems, Neural Comput. Appl. 27 (4) (2016) 1053–1073.

[21] A.A. Heidari, S. Mirjalili, H. Faris, I. Aljarah, M. Mafarja, H. Chen, Harris hawks optimization: Algorithm and applications, Future Gener. Comput. Syst. 97 (2019) 849–872.

[22] S. Mirjalili, Moth-flame optimization algorithm: A novel nature-inspired heuristic paradigm, Knowl.-Based Syst. 89 (2015) 228–249.

[23] S. Mirjalili, A. Lewis, The whale optimization algorithm, Adv. Eng. Softw. 95 (2016) 51–67.

[24] H. Shayanfar, F.S. Gharehchopogh, Farmland fertility: A new metaheuristic algorithm for solving continuous optimization problems, Appl. Soft Comput. 71 (2018) 728–746.

[25] A.I. Wagan, M.M. Shaikh, A new metaheuristic optimization algorithm inspired by human dynasties with an application to the wind turbine micrositing problem, Appl. Soft Comput. 90 (2020) 106176.

[26] M.H. Sulaiman, Z. Mustaffa, M.M. Saari, H. Daniyal, Barnacles mating optimizer: A new bio-inspired algorithm for solving engineering optimization problems, Eng. Appl. Artif. Intell. 87 (2020) 103330.

[27] A. Faramarzi, M. Heidarinejad, S. Mirjalili, A.H. Gandomi, Marine predators algorithm: A nature-inspired metaheuristic, Expert Syst. Appl. (2020) 113377.

[28] S.H.S. Moosavi, V.K. Bardsiri, Poor and rich optimization algorithm: A new human-based and multi populations algorithm, Eng. Appl. Artif. Intell. 86 (2019) 165–181.

[29] L. Wang, Y. Xiong, S. Li, Y.-R. Zeng, New fruit fly optimization algorithm with joint search strategies for function optimization problems, Knowl.-Based Syst. 176 (2019) 77–96.

[30] S. Mirjalili, Ant colony optimisation, in: Evolutionary Algorithms and Neural Networks. Studies in Computational Intelligence, Springer, Cham, 2019, pp. 33–42, http://dx.doi.org/10.1007/978-3-319-93025-1_3.

[31] J.C. Bansal, H. Sharma, S.S. Jadon, M. Clerc, Spider monkey optimization algorithm for numerical optimization, Memetic Comput. 6 (1) (2014) 31–47.

[32] H.N. Ghafil, K. Jármai, Optimization for Robot Modelling with MATLAB, Springer, 2020.

[33] R. Eberhart, J. Kennedy, A new optimizer using particle swarm theory, in: MHS'95. Proceedings of the Sixth International Symposium on Micro Machine and Human Science, IEEE, 1995, pp. 39–43.

[34] M. Kohler, M.M. Vellasco, R. Tanscheit, PSO+: A new particle swarm optimization algorithm for constrained problems, Appl. Soft Comput. 85 (2019) 105865.

[35] S. Kaur, L.K. Awasthi, A. Sangal, G. Dhiman, Tunicate swarm algorithm: A new bio-inspired based metaheuristic paradigm for global optimization, Eng. Appl. Artif. Intell. 90 (2020) 103541.

[36] S. Mirjalili, A.H. Gandomi, S.Z. Mirjalili, S. Saremi, H. Faris, S.M. Mirjalili, Salp swarm algorithm: A bio-inspired optimizer for engineering design problems, Adv. Eng. Softw. 114 (2017) 163–191.

[37] S. Saremi, S. Mirjalili, A. Lewis, Grasshopper optimisation algorithm: theory and application, Adv. Eng. Softw. 105 (2017) 30–47.

[38] H. Yapici, N. Cetinkaya, A new meta-heuristic optimizer: Pathfinder algorithm, Appl. Soft Comput. 78 (2019) 545–568.

[39] J.H. Holland, Adaptation in Natural and Artificial Systems: An Introductory Analysis with Applications To Biology, Control, and Artificial Intelligence, MIT Press, Google books, 1992.

[40] R. Storn, K. Price, Differential evolution–a simple and efficient heuristic for global optimization over continuous spaces, J. Glob. Optim. 11 (4) (1997) 341–359.

[41] D. Karaboğa, S. Ökdem, A simple and global optimization algorithm for engineering problems: differential evolution algorithm, Turk. J. Electr. Eng. Comput. Sci. 12 (1) (2004) 53–60.

[42] M. Pelikan, D.E. Goldberg, E. Cantú-Paz, BOA: The Bayesian optimization algorithm, in: Proceedings of the genetic and evolutionary computation conference GECCO-99, 1999, pp. 525–532.

[43] N. Hansen, S.D. Müller, P. Koumoutsakos, Reducing the time complexity of the derandomized evolution strategy with covariance matrix adaptation (CMA-ES), Evol. Comput. 11 (1) (2003) 1–18.

[44] A.H. Kashan, R. Tavakkoli-Moghaddam, M. Gen, Find-Fix-Finish-Exploit-Analyze (F3EA) meta-heuristic algorithm: An effective algorithm with new evolutionary operators for global optimization, Comput. Ind. Eng. 128 (2019) 192–218.

[45] M. Mafarja, I. Aljarah, A.A. Heidari, A.I. Hammouri, H. Faris, M.A.-Z. Ala', S. Mirjalili, Evolutionary population dynamics and grasshopper optimization approaches for feature selection problems, Knowl.-Based Syst. 145 (2018) 25–45.

[46] S.S. Jadon, R. Tiwari, H. Sharma, J.C. Bansal, Hybrid artificial bee colony algorithm with differential evolution, Appl. Soft Comput. 58 (2017) 11–24.

[47] F.J. Solis, R.J.-B. Wets, Minimization by random search techniques, Math. Oper. Res. 6 (1) (1981) 19–30.

[48] D. Wolpert, No free lunch theorem for optimization, IEEE Trans. Evol. Comput. (1) (1997) 467–482.

[49] S. Kirkpatrick, Optimization by simulated annealing: Quantitative studies, J. Stat. Phys. 34 (5–6) (1984) 975–986.

[50] P.P. Rebouças Filho, S.P.P. da Silva, V.N. Praxedes, J. Hemanth, V.H.C. de Albuquerque, Control of singularity trajectory tracking for robotic manipulator by genetic algorithms, J. Comput. Sci. 30 (2019) 55–64.

[51] Z. Li, C. Hu, C. Ding, G. Liu, B. He, Stochastic gradient particle swarm optimization based entry trajectory rapid planning for hypersonic glide vehicles, Aerosp. Sci. Technol. 76 (2018) 176–186.

[52] V. Acharya, K.V. Kumar, B. Gopalsamy, B. Suresh, Genetic algorithm based kinematic synthesis of an eight bar flap deployment mechanism in a typical transport aircraft, Mater. Today: Proc. 5 (11) (2018) 24887–24897.

[53] V.-H. Truong, S.-E. Kim, Reliability-based design optimization of nonlinear inelastic trusses using improved differential evolution algorithm, Adv. Eng. Softw. 121 (2018) 59–74.

[54] J.P. George Lindfield, Optimization methods, in: J.P. George Lindfield (Ed.), Numerical Methods, fourth ed., Academic Press, 2019, pp. 433–483.

[55] A. Mortazavi, V. Toğan, M. Moloodpoor, Solution of structural and mathematical optimization problems using a new hybrid swarm intelligence optimization algorithm, Adv. Eng. Softw. 127 (2019) 106–123.

[56] M. Farshchin, M. Maniat, C.V. Camp, S. Pezeshk, School based optimization algorithm for design of steel frames, Eng. Struct. 171 (2018) 326–335.

[57] J. Jung, J.I. Yoon, H.K. Park, J.Y. Kim, H.S. Kim, Bayesian Approach in predicting mechanical properties of materials: Application to dual phase steels, Mater. Sci. Eng. A 743 (2019) 382–390.

[58] D. Ke, X. Liu, Y. Zhi, X. Hu, L. Liu, Experiment on properties differentiation in tailor rolled blank of dual phase steel, Mater. Sci. Eng. A 742 (2019) 629–635.

[59] H. Ghafil, K. Jármai, Comparative study of particle swarm optimization and artificial bee colony algorithms, 2018.

[60] H. Ghafil, Inverse acceleration solution for robot manipulators using harmony search algorithm, Int. J. Comput. Appl. 6 (114) (2016) 1–7.

[61] L. Bottarelli, M. Bicego, J. Blum, A. Farinelli, Orienteering-based informative path planning for environmental monitoring, Eng. Appl. Artif. Intell. 77 (2019) 46–58.

[62] F. Ropero, P. Muñoz, M.D. R.-M.oreno, TERRA: A path planning algorithm for cooperative UGV–UAV exploration, Eng. Appl. Artif. Intell. 78 (2019) 260–272.

[63] N. Metropolis, A.W. Rosenbluth, M.N. Rosenbluth, A.H. Teller, E. Teller, Equation of state calculations by fast computing machines, J. Chem. Phys. 21 (6) (1953) 1087–1092.

[64] A. El Afia, M. Lalaoui, R. Chiheb, A self controlled simulated annealing algorithm using hidden markov model state classification, Procedia Comput. Sci. 148 (2019) 512–521.

[65] K. Dorgham, I. Nouaouri, H. Ben-Romdhane, S. Krichen, A hybrid simulated annealing approach for the patient bed assignment problem, Procedia Comput. Sci. 159 (2019) 408–417.

[66] Z. Issam, M. Al-Omani, K. Aldhafeeri, A new approach based on the hybridization of simulated annealing algorithm and tabu search to solve the static ambulance routing problem, Procedia Comput. Sci. 159 (2019) 1216–1228.

[67] A. Askarzadeh, L. dos Santos Coelho, C.E. Klein, V.C. Mariani, A population-based simulated annealing algorithm for global optimization, in: 2016 IEEE International Conference on Systems, Man, and Cybernetics, SMC, IEEE, 2016, pp. 004626–004633.

[68] F. Martinez-Rios, A new hybridized algorithm based on population-based simulated annealing with an experimental study of phase transition in 3-SAT, Procedia Comput. Sci. 116 (2017) 427–434.

[69] N. Sadati, M. Zamani, H.R.F. Mahdavian, Hybrid particle swarm-based-simulated annealing optimization techniques, in: IECON 2006-32nd Annual Conference on IEEE Industrial Electronics, IEEE, 2006, pp. 644–648.

[70] B. Morales-Castañeda, D. Zaldívar, E. Cuevas, O. Maciel-Castillo, I. Aranguren, F. Fausto, An improved simulated annealing algorithm based on ancient metallurgy techniques, Appl. Soft Comput. 84 (2019) 105761.

[71] A.A. Kida, A.E.L. Rivas, L.A. Gallego, An improved simulated annealing–linear programming hybrid algorithm applied to the optimal coordination of directional overcurrent relays, Electr. Power Syst. Res. 181 (2020) 106197.

[72] M.C. Aguitoni, L.V. Pavão, M.A.d.S.S. Ravagnani, Heat exchanger network synthesis combining simulated annealing and differential evolution, Energy 181 (2019) 654–664.

[73] C. Suppan, T. Hebesberger, A. Pichler, J. Rehrl, O. Kolednik, On the microstructure control of the bendability of advanced high strength steels, Mater. Sci. Eng. A 735 (2018) 89–98.

[74] J. Zhao, Z. Jiang, Thermomechanical processing of advanced high strength steels, Prog. Mater. Sci. (2018).

[75] J.-H. Schmitt, T. Iung, New developments of advanced high-strength steels for automotive applications, C. R. Phys. 19 (8) (2018) 641–656.

[76] W. Sun, Y. Wu, S. Yang, C. Hutchinson, Advanced high strength steel (AHSS) development through chemical patterning of austenite, Scr. Mater. 146 (2018) 60–63.

[77] Y.M. Kim, S.K. Kim, N.J. Kim, Simple method for tailoring the optimum microstructures of high-strength low-alloyed steels by the use of constitutive equation, Mater. Sci. Eng. A 743 (2019) 138–147.

[78] T. Dutta, S. Dey, S. Datta, D. Das, Designing dual-phase steels with improved performance using ANN and GA in tandem, Comput. Mater. Sci. 157 (2019) 6–16.

[79] H.N. Ghafil, K. Jármai, Research and application of industrial robot manipulators in vehicle and automotive engineering, a survey, in: Vehicle and Automotive Engineering, Springer, 2018, pp. 611–623, http://dx.doi.org/10.1007/978-3-319-75677-6_53.

[80] J. Liang, B. Qu, P. Suganthan, Q. Chen, Problem Definitions and Evaluation Criteria for the CEC 2015 Competition on Learning-Based Real-Parameter Single Objective Optimization, Vol. 29, Technical Report201411A, Computational Intelligence Laboratory, Zhengzhou University, Zhengzhou China and Technical Report, Nanyang Technological University, Singapore, 2014, pp. 625–640.

[81] I.B. Aydilek, A hybrid firefly and particle swarm optimization algorithm for computationally expensive numerical problems, Appl. Soft Comput. 66 (2018) 232–249.

[82] X.-S. Yang, X. He, Firefly algorithm: recent advances and applications, 2013, arXiv preprint arXiv:1308.3898.

[83] P. Kora, K.S.R. Krishna, Hybrid firefly and particle swarm optimization algorithm for the detection of bundle branch block, Int. J. Cardiovasc. Acad. 2 (1) (2016) 44–48.

[84] S.B.D. Surjanovic, Virtual Library of Simulation Experiments: Test Functions and Datasets, 2013, http://www.sfu.ca/~ssurjano, (Accessed 23 November 2018).

[85] M. Molga, C. Smutnicki, Test Functions for Optimization Needs, vol. 101, 2005.

[86] M. Jain, V. Singh, A. Rani, A novel nature-inspired algorithm for optimization: Squirrel search algorithm, Swarm Evol. Comput. 44 (2019) 148–175.

[87] H.A. Kasdirin, N. Yahya, M. Aras, M. Tokhi, Hybridizing invasive weed optimization with firefly algorithm for unconstrained and constrained optimization problems, J. Theor. Appl. Inf. Technol. 95 (4) (2017) 912–927.

[88] A.R. Mehrabian, C. Lucas, A novel numerical optimization algorithm inspired from weed colonization, Ecol. Inform. 1 (4) (2006) 355–366.

[89] H.N. Ghafil, Optimum Path Planning and Performance Analysis of a Robot Manipulator, Al Nahrain University, 2013.

[90] A. Gasparetto, V. Zanotto, Optimal trajectory planning for industrial robots, Adv. Eng. Softw. 41 (4) (2010) 548–556.

[91] B. Li, Z. Shao, Simultaneous dynamic optimization: A trajectory planning method for nonholonomic car-like robots, Adv. Eng. Softw. 87 (2015) 30–42.

[92] G.H. Behforooz, A comparison of thee (3) and not-a-knot cubic splines, Appl. Math. Comput. 72 (2–3) (1995) 219–223.

[93] Mathworks, Documentation, 2019, https://ch.mathworks.com/help/matlab/ref/spline.html, (Accessed 16 April 2019).

[94] S. Mirjalili, Website, 2019, http://www.alimirjalili.com/index.html, (Accessed 15 April 2019).

[95] A. Askarzadeh, A novel metaheuristic method for solving constrained engineering optimization problems: Crow search algorithm, Comput. Struct. 169 (2016) 1–12.

[96] C.A. Coello, An updated survey of GA-based multiobjective optimization techniques, ACM Comput. Surv. 32 (2) (2000) 109–143.

[97] C.A.C. Coello, E.M. Montes, Constraint-handling in genetic algorithms through the use of dominance-based tournament selection, Adv. Eng. Inform. 16 (3) (2002) 193–203.

[98] Q. He, L. Wang, An effective co-evolutionary particle swarm optimization for constrained engineering design problems, Eng. Appl. Artif. Intell. 20 (1) (2007) 89–99.

[99] J. Yang, Y. Jin, Hierarchy particle swarm optimization algorithm (HPSO) and its application in multi-objective operation of hydropower stations, in: 2011 3rd International Workshop on Intelligent Systems and Applications, IEEE, 2011, pp. 1–4.
[100] L. dos Santos Coelho, GaussIan quantum-behaved particle swarm optimization approaches for constrained engineering design problems, Expert Syst. Appl. 37 (2) (2010) 1676–1683.
[101] M. Zhang, W. Luo, X. Wang, Differential evolution with dynamic stochastic selection for constrained optimization, Inform. Sci. 178 (15) (2008) 3043–3074.
[102] H. Liu, Z. Cai, Y. Wang, Hybridizing particle swarm optimization with differential evolution for constrained numerical and engineering optimization, Appl. Soft Comput. 10 (2) (2010) 629–640.
[103] T. Ray, K.-M. Liew, Society and civilization: An optimization algorithm based on the simulation of social behavior, IEEE Trans. Evol. Comput. 7 (4) (2003) 386–396.
[104] K.E. Parsopoulos, M.N. Vrahatis, Unified particle swarm optimization for solving constrained engineering optimization problems, in: International Conference on Natural Computation, Springer, 2005, pp. 582–591.
[105] E. Mezura-Montes, C.A.C. Coello, A simple multimembered evolution strategy to solve constrained optimization problems, IEEE Transactions on Evolutionary computation 9 (1) (2005) 1–17.
[106] B. Thamaraikannan, V. Thirunavukkarasu, Design optimization of mechanical components using an enhanced teaching-learning based optimization algorithm with differential operator, Math. Probl. Eng. 2014 (2014).
[107] R.V. Rao, V.J. Savsani, D. Vakharia, Teaching–learning-based optimization: a novel method for constrained mechanical design optimization problems, Comput. Aided Des. 43 (3) (2011) 303–315.
[108] A. Sadollah, A. Bahreininejad, H. Eskandar, M. Hamdi, Mine blast algorithm: A new population based algorithm for solving constrained engineering optimization problems, Appl. Soft Comput. 13 (5) (2013) 2592–2612.
[109] Mathworks, MATLAB Answers, 2017, http://www.mathworks.com/matlabcentral/answers/374578-why-does-runtime-decrease-the-more-times-i-execute-a-command, (Accessed 20 April 2020).
[110] C. Dojo, Introduction to big o notation and time complexity (data structures & algorithms #7), 2018, Youtube Vedio. http://www.youtube.com/watch?v=D6xkbGLQesk. (Accessed 20 April 2020).