Applied Soft Computing 49 (2016) 498–509

Contents lists available at ScienceDirect

Applied Soft Computing

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

Nearest neighbour cuckoo search algorithm with probabilistic mutation

Lijin Wang [a,b], Yiwen Zhong [a], Yilong Yin [b,c,*]

[a] College of Computer and Information Science, Fujian Agriculture and Forestry University, Fuzhou 350002, PR China
[b] School of Computer Science and Technology, Shandong University, Jinan 250101, PR China
[c] School of Computer Science and Technology, Shandong University of Finance and Economics, Jinan 250101, PR China

ARTICLE INFO

Article history:
Received 28 April 2014
Received in revised form 18 August 2015
Accepted 12 August 2016
Available online 2 September 2016

Keywords:
Cuckoo search algorithm
Nearest neighbour
Solution-based similar metric
Fitness-based similar metric
Probabilistic mutation

ABSTRACT

In this study, we present a nearest neighbour cuckoo search algorithm with probabilistic mutation, called NNCS. In the proposed approach, the nearest neighbour strategy is utilized to select guides to search for new solutions by using the nearest neighbour solutions instead of the best solution obtained so far. In the proposed strategy, we respectively employ a solution-based and a fitness-based similar metrics to select the nearest neighbour solutions for implementation. Furthermore, the probabilistic mutation strategy is used to control the new solutions learn from the nearest neighbour ones in partial dimensions only. In addition, the nearest neighbour strategy helps the best solution participate in searching too. Extensive experiments, which are carried on 20 benchmark functions with different properties, demonstrate the improvement in effectiveness and efficiency of the nearest neighbour strategy and the probabilistic mutation strategy.

1. Introduction

Optimization has played an important role on decision science and on the analysis of physical system. Optimization problems can be mathematically formulated as the minimization or maximization of objective functions subject to constraints on their variables. Without loss of generality, a (continuous) optimization problem can be written as follow:

$$\begin{aligned} &\text{minimize} \quad f(\vec{x}), \vec{x} = (x_1, x_2, \ldots, x_n) \in R^n \\ &\text{subject to} \quad g_i(\vec{x}) \le 0, \quad i \in \varepsilon \\ &\qquad\qquad h_j(\vec{x}) = 0, \quad j \in \xi \end{aligned} \tag{1}$$

where $f(\vec{x})$ is the objective function to be minimized over the vector $\vec{x}$ representing the decision variables of the problem, $g_i(\vec{x})$ is the ith inequality constraints, and $h_j(\vec{x})$ is the jth equality constraints; ξ and ε are sets of indices for equality and inequality constraints, respectively.

Since nature-inspired meta-heuristic algorithms have shown some promising results in solving tough optimization problems, like the design of fuzzy control systems [1], performance optimization of electrical drives [2] and bi-level programming problem [3], they have received much attention and huge popularity recently. These approaches include genetic algorithm (GA) [4], differential evolution (DE) [5], ant colony optimization (ACO) [6], particle swarm optimization (PSO) [7,8], artificial bee colony (ABC) [9], social emotion optimization algorithm (SEOA) [10,11], bat algorithm (BA) [12], firefly algorithm (FA) [13], harmony search (HS) [14], biogeography-based optimization (BBO) [15], group search optimizer (GSO) [16], and their hybridizations, such as DE/BBO [17], FPSO + FGA [18], and so on.

Cuckoo search (CS) [19,20] is another nature-inspired meta-heuristic algorithm based on the obligate brood parasitic behavior of some cuckoo species in combination with the Lévy flights behavior of some birds and fruit files. CS is a simple yet very promising optimization technique. At each iteration process, CS firstly uses Lévy flights random walk (LFRW) to search for new solutions. Secondly, CS continues to generate new solutions in terms of biased/selective random walk (BSRW) which employs a crossover operator. After each random walk, a greedy strategy is used to select a better solution from the current and new generated solutions according to their fitness.

Due to simple concepts, easy to implement and few parameters, CS has been successfully applied in a wide variety of problems from diverse fields. Some works have focused on random walk step-size for function optimization problems [21–24]. Some attempts have applied some other evolution algorithms to improve the

* Corresponding author at: School of Computer Science and Technology, Shandong University, Jinan 250101, PR China.
E-mail address: ylyin@sdu.edu.cn (Y. Yin).

http://dx.doi.org/10.1016/j.asoc.2016.08.021
1568-4946/

performance of CS for complex optimization problems [25–34]. Others have been proposed to deal with combinatorial optimization problems [35–38], and multi-object optimization problems [39,40].

Nevertheless, there is no specific algorithm to achieve the best solution for all optimization problems [41]. Some may provide better solutions and faster convergence for particular optimization problems than others. Therefore, it seems necessary to propose new improved or enhanced algorithms to improve the quality of solutions and convergence speed. On the other hand, in LFRW, CS searches for new solutions based on the best solution obtained so far. In this case, the solutions may easily be attracted to the best solution region because these solutions learn from the best one, and the convergence speed may be fast. However, if the current best solution is far away from the global optimum, and if the search environment is complex with numerous local optima, the solutions may get trapped in a local optimum. Moreover, according to the implementation of conventional CS, the best solution cannot achieve a new solution because the step size is zero. In this paper, we propose NNCS, a nearest neighbour cuckoo search algorithm with probabilistic mutation, to accelerate convergence speed and improve the solutions quality for continuous function optimization problems. NNCS introduces the nearest neighbour strategy and the probabilistic mutation strategy to remedy this weakness of LFRW. The nearest neighbour strategy makes the new generated solutions be around their nearest neighbour ones instead of the best solution obtained so far. In the proposed strategy, we employ a solution-based and a fitness-based similar metrics to select the nearest neighbour solutions for implementation, respectively. The probabilistic mutation strategy lets the solutions learn from the nearest neighbour solutions in partial dimensions only. In addition, the nearest neighbour strategy can help the best solution participate in searching instead of doing nothing in the implementation of the conventional CS. Simulations and comparisons are carried on 20 benchmark functions with uni-modal, multi-modal, rotated and shifted properties. The results indicate the improvement in effectiveness and efficiency of the nearest neighbour strategy and the probability mutation strategy, and show the proposed algorithm is competitive for dealing with optimization problems compared with other algorithms.

The remainder of this paper is organized as follows. Section 2 describes the standard cuckoo search algorithm. Section 3 presents the NNCS algorithm. Section 4 reports the experimental results. Section 5 concludes this paper.

2. Cuckoo search algorithm

CS, developed recently by Yang and Deb [19,20], is a simple yet very promising population-based stochastic search technique. For the sake of simplicity, we consider that CS is used to solve an unconstrained optimization problem by minimizing $f(X)$ with the solution space $[x_{j,min}, x_{j,max}]$, $j = 1, 2, \ldots, D$. In CS, a nest represents a candidate solution $X = (x_{1,\ldots}, x_D)$, and the pseudo-code of CS is shown in Algorithm 1.

Algorithm 1. CS

```
G ← 1;
X_G = (X_{i,G}, ..., X_{NP,G}) ← Initialize solutions;
Fitness ← Evaluate X_G;
FES ← NP;
BestX ← Obtain the best solution so far;
while stop condition does not meet do
  for i ← 1 to NP do
    U_{i,G} ← Search for a new solution using LFRW;
    Perform the boundary-handling method;
    X_{i,G} ← Greedily select a better solution from U_{i,G} and X_{i,G} according their fitness;
    FES ← FES+1;
  end for
  for i ← 1 to NP do
    U_{i,G} ← Search for a new solution using BSRW;
    Perform the boundary-handling method;
    X_{i,G+1} ← Greedily select a better solution from U_{i,G} and X_{i,G} according their fitness;
    FES ← FES+1;
  end for
  BestX ← Obtain the best solution so far;
  G ← G+1;
end while
Output the best solution.
```

In the initialization phase, CS initializes solutions that are randomly sampled from solution space by

$$x_{i,j,0} = x_{i,j,\min} + r(x_{i,j,\max} - x_{i,j,\min}), \quad i = 1, 2, \ldots, NP \tag{2}$$

where r represents a uniformly distributed random variable with the range [0,1], and NP is the population size.

After initialization, CS goes into an iterative phase where two random walks, LFRW and BSRW, are employed to search for new solutions.

LFRW is a random walk whose step size is drawn from a Lévy distribution. At generation $G(G > 0)$, LFRW can be formulated as follows:

$$U_{i,G} = X_{i,G} + \alpha \oplus \text{Lé vy}(\beta) \tag{3}$$

where $\alpha > 0$ is a step size, which should be related to the scales of the problem of interest [20], the product $\oplus$ means entry-wise multiplications, $U_{i,G}$ is the new solution of $X_{i,G}$, and Lévy(β) can be written in terms of a simple power-law formula

$$\text{Lé vy}(\beta) \sim t^{-1-\beta}, \quad 0 < \beta \le 2 \tag{4}$$

where t is a random variable, and β is a stability index. It is worthy of noting that the well-known Gaussian and Cauchy distribution are its special cases when its stability index β is 2 and 1, respectively.

In implementation, Lévy(β) can be calculated as follow [20]:

$$\text{Lé vy}(\beta) \sim \frac{\phi \times \mu}{|\nu|^{1/\beta}} \tag{5}$$

$$\phi = \left[\frac{\Gamma(1+\beta) \times \sin(\frac{\pi \times \beta}{2})}{\Gamma\left(\frac{1+\beta}{2}\right) \times \beta \times 2^{\frac{\beta-1}{2}}} \right]^{1/\beta} \tag{6}$$

where β is suggested as 1.5 in [20], μ and ν are random numbers drawn from a normal distribution with mean of 0 and standard deviation of 1, and Γ is *gamma function*.

In CS, LFRW searches for new solutions around the best solution obtained so far, and Eq. (3) can be reformulated as

$$U_{i,G} = X_{i,G} + \alpha_0 \frac{\phi \times \mu}{|\nu|^{1/\beta}} (X_{i,G} - X_{best}) \tag{7}$$

where α_0 is a scaling factor (generally, α_0=0.01), and X_{best} represents the best solution obtained so far.

In LFRW, for the solution $X_{i,G}$, CS searches for a new solution $U_{i,G}$ around the best solution obtained so far. After LFRW, CS uses a greedy strategy to select the better solution as the solution $X_{i,G}$ from $X_{i,G}$ and $U_{i,G}$ according to their fitness. In the case of the greedy strategy, CS can be easy trapped into a local optimum, therefore, BSRW is used to discover new solutions far enough away from the current best solution region by far field randomization [19]. Firstly, a trial solution $V_{i,G}$ is built with a mutation of the solution $X_{i,G}$ as base vector and two randomly selected solutions as perturbed vectors, as seen in Eq. (8)

$$V_{i,G} = X_{i,G} + r \times (X_{m,G} - X_{n,G}) \tag{8}$$

where, m and n are random indexes, r is a scaling factor with the range [0,1], and p_a is a fraction probability. Secondly, a new solution

$U_{i,G}$ is generated by a crossover operator from $X_{i,G}$ and the trial solution $V_{i,G}$, as follows:

$$u_{i,j,G} = \begin{cases} v_{i,j,G}, & rand > p_a \\ x_{i,j,G}, & \text{otherwise} \end{cases} \tag{9}$$

After BSRW, by using the greedy strategy, the next generation solution $X_{i,G+1}$ is selected from $X_{i,G}$ and $U_{i,G}$ according to their fitness. At the end of each iteration process, the best solution obtained so far is updated.

Note that CS utilizes a boundary-handling method to ensure the generated new solutions by using LFRW or BSRW not to be out of solution space. In this paper, to make a fair comparison, we use the same yet simple boundary-handling method for all mentioned algorithms. The detailed description will be given in Eq. Eq. (15). It is worth pointing out that CS can also be utilized to solve constrained optimization problems by using constraint-handling techniques. The literature [42] proposed an effective hybrid cuckoo search algorithm for constrained optimization that relies on Augmented Lagrangian method for constraint-handling.

3. Nearest neighbour cuckoo search algorithm with probabilistic mutation

In LFRW, CS searches for new solutions based on the best solution obtained so far. In this case, all solutions interact with the best solution. In fact, Ballerini et al., have discovered that the interaction among individuals depends on the neighbour distance with a fixed number of neighbours [43]. It suggests that not all individuals interact with the best one, but with their nearest neighbour ones. This motivates us to make solutions to interact with their nearest neighbour solutions.

Accordingly, we combine the nearest neighbour strategy with CS. The nearest neighbour strategy is used to select nearest neighbour solutions instead of the best solution for the solutions interacting in LFRW. The nearest neighbour strategy is a variant scheme of the nearest neighbour search algorithm which is an optimization problem for finding closest (or most similar) points. Mathematically, the nearest neighbour search problem is defined as follows: [44].

$$X_{nearest} = arg\min\{d(x, y)\}, \quad x \in S, \quad y \in M \tag{10}$$

where, M is a metric space, S is a set of points in M, y is a query point, $X_{nearest}$ is the closest point in S to y, and $d(x, y)$ presents the similar metric. In this case, we rewrite Eq. (7) as follows:

$$U_{i,G} = X_{i,G} + r\frac{\phi \times \mu}{|\nu|^{1/\beta}}(X_{i,G} - X_{i,G,nearest}) \tag{11}$$

where r presents a varied scaling factor drawn a uniformly distribution in the interval of [0,1], and $X_{i,G,nearest}$ is the nearest neighbour solution in swam of $X_{i,G}$.

In NNCS, we employ a solution-based and a fitness-based similar metrics $d(x, y)$ for implementation, resulting in two NNCS algorithms, NNCS-S and NNCS-F, respectively.

The solution-based similar metric can be measured using the Euclidean distance, Manhattan distance, or other distance metric. In this paper, the default distance metric is Euclidean distance, which between any two D-dimensional vectors X_i and X_j is given as follows:

$$d(X_i, Y_i) = \sqrt{\sum_{d=1}^{D}(X_{i,d} - Y_{i,d})^2} \tag{12}$$

In addition, the fitness-based similar metric between any two D-dimensional vectors X_i and X_j can be given as follows:

$$d(X_i, Y_i) = \|f(X_i) - f(Y_i)\|_2 \tag{13}$$

where $f(X)$ is an objective function, and $\|\cdot\|_2$ represents the 2-norm.

According to Eq. (11), all solutions learn from their nearest neighbour ones with all dimensions. However, the nearest neighbour solutions are not the global optimum, which may have good values on some dimensions and bad values on others. This will results in that some dimensions of new solutions may have bad values that are deteriorated by bad values of the nearest neighbour solutions. Therefore, the probabilistic mutation strategy is employed to control that the solutions learn from the nearest neighbour ones in partial dimensions only. The probabilistic mutation strategy can be defined as follows:

$$u_{i,j,G} = \begin{cases} x_{i,j,G} + r\frac{\phi \times \mu}{|\nu|^{1/\beta}}(x_{i,j,G} - x_{i,j,G,nearest}), & rand < p \\ x_{i,j,G}, & \text{otherwise} \end{cases} \tag{14}$$

where, *rand* is a random number drawn a uniformly distribution in the interval of [0,1], and p is a constant. In addition, a simple boundary-handing method, as similar done in [45], works as follows: if the jth element $u_{i,j,G}$ is out of the solution space $[x_{j,min}, x_{j,max}]$, then $u_{i,j,G}$ is reset as follows:

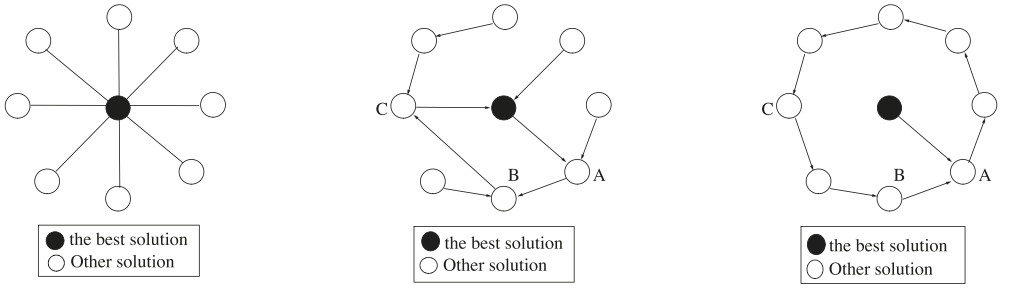
$$u_{i,j,G} = \begin{cases} \min\{x_{j,\max}, 2x_{j,\min} - u_{i,j,G}\}, & u_{i,j,G} < x_{j,\min} \\ \max\{x_{j,\min}, 2x_{j,\max} - u_{i,j,G}\}, & u_{i,j,G} > x_{j,\max} \end{cases} \tag{15}$$

According to the above descriptions, we summarize them and present the pseudo-code of NNCS as shown in Algorithm 2.

Algorithm 2. NNCS

```
G ← 1;
X_G = (X_{i,G}, ..., X_{NP,G}) ← Initialize solutions;
Fitness ← Evaluate X_G;
FES ← NP;
BestX ← Obtain the best solution so far;
while stop condition does not meet do
  for i ← 1 to NP do
    X_{i,G,nearest} ← Search the nearest neighbour using Eq. (10), Eq. (12) or Eq. (13);
    U_{i,G} ← Search for a new solution using Eq. (14);
    Perform the boundary-handling method using Eq. (15);
    X_{i,G} ← Greedily select a better solution from U_{i,G} and X_{i,G} according to their fitness;
    FES ← FES+1;
  end for
  for i ← 1 to NP do
    U_{i,G} ← Search for a new solution using BSRW;
    Perform the boundary-handling method using Eq. (15);
    X_{i,G+1} ← Greedily select a better solution from U_{i,G} and X_{i,G} according to their fitness;
    FES ← FES+1;
  end for
  BestX ← Obtain the best solution so far;
  G ← G+1;
end while
Output the best solution.
```

It is worthy of pointing out that the nearest neighbour strategy is overall different from the similar strategies applied in PSO community [46,47]. The neighbourhood topology of LFRW in CS is a wheel topology that is similar in [46,47], as seen in Fig. 1(a), while that of LFRW in NNCS is a non-wheel topology overall, as shown in Fig. 1(b). Observed from Fig. 1(b), solution "A" and solution "B" respectively search for new solutions based on their nearest solutions "B" and "C" instead of the best one, resulting in improving the population diversity. Moreover, the best solution takes part in the exploration by learning from "A". In addition, except for the best solution, if the nearest neighbour solution of each individual is different, the



(a) wheel topology (b) non-wheel topology (c) similar ring topology

Fig. 1. Neighbourhood topology of LFRW in CS and in NNCS.

neighbourhood topology of LFRW in NNCS can be regarded as a particularly similar ring topology, as seen in Fig. 1 (c). Note that the probabilistic mutation strategy in NNCS is different from the mutation strategy in CMA-ES [48]. The former is randomized and non-adaptive, while the latter is derandomized and self-adaptive.

4. Experimental verifications

In this section, in order to validate the performance of NNCS, we carry out different experiments on a suite of 20 unconstrained single-objective benchmark functions which are described in Appendix A. These functions are chosen from the literatures [49,50], including unimodal functions F_{sph}, and F_{ros}, multimodal functions F_{ack}, F_{grw}, F_{ras}, F_{sch}, F_{sal}, F_{wht}, F_{pn1} and F_{pn2}, and rotated and/or shifted functions F_1–F_{10}. More detailed description of them can be found in the literatures [49,50].

For NNCS, there are three control parameters, namely, the population size NP, the fraction probability p_a, and the mutation probability p. For all experiments, unless a change is mentioned, the population size NP is D which is the dimension of problem, as similar done in [50], the fraction probability p_a is 0.25 which is suggested in [19], and the mutation probability p is 0.25. More discussions about the settings of p can be found in Section 4.5. Moreover, in our experiments, each algorithm is used to optimize each benchmark function over 50 independent runs. The maximum number of function evaluations (MaxFEs) is $10{,}000\times D$.

In our experimental studies, we select four performance criteria from the literatures [49,50] to evaluate the performance of the algorithms.

Error: Error is the function error value defined as $(f(x)-f(x^*))$, where x^* is the known global optimum of the function, and x is the best solution obtained by the algorithm in a run. The minimum function error value that each algorithm can find is recorded in different runs, and the average and the standard deviation of the error value are calculated. The notation $AVG_{Er} \pm SD_{Er}$ is used in different tables.

Number of function evaluations (FES): The number of function evaluations is also recorded in different runs when each algorithm achieves the value to reach (VTR) suggested in [50] within MaxFEs. The average and standard deviation of FES, denoted $AVG_{Ev} \pm SD_{Ev}$ is used different tables.

Number of successful runs (SR): The number of successful runs is recorded when the VTR is reached within MaxFEs.

Convergence graphs: The convergence graphs show the average function error value performance of the total runs in respective experiments.

Additionally, the Wilcoxon signed-rank test at the 5% significant level is used to show significance between two algorithms [51]. The "‡" symbol shows the null hypothesis is rejected, and the first algorithm outperforms the second one. The "†" symbol means the null hypothesis is rejected, and the first algorithm is inferior to the second one. The "=" symbol reveals the null hypothesis is accepted, and the first algorithm ties the second one. The total number of each symbol is summarized at the bottom of different table. Meanwhile, the Friedman test, as similar done in [52], is employed to show the competitiveness of the compared algorithms.

4.1. Solution accuracy study

To show how NNCS can improve solution accuracy, we carry out experiments on the 20 benchmark functions at $D = 30$. The solutions accuracy obtained by NNCS is compared with the ones obtained by CS, shown in Table 1. Compared to the solutions accuracy obtained by CS, the higher accurate solutions gained by NNCS are marked in boldface. Moreover, the results of the Wilcoxon signed-rank test between CS and NNCS-S, and NNCS-F, are also given, respectively.

From Table 1, we can see that in terms of solutions accuracy, NNCS-F and NNCS-S outperform CS on the unimodal functions, and the proposed strategies bring solutions with much higher accuracy to the problems. For example, for very simple unimodal function F_{sph}, NNCS-F and NNCS-S provide solutions with higher quality than those provided by CS. Moreover, NNCS-F and NNCS-S are much better on more complex F_{ros}. It can be also seen that NNCS-F and NNCS-S surpass CS on all 8 multimodal functions according to solutions accuracy. NNCS-F and NNCS-S obtain the global optimum on the function F_{grw}, while CS doesn't. Furthermore, NNCS-F and NNCS-S bring solutions with much higher accuracy on other multimodal functions, especially on the functions F_{pn1} and F_{pn2}. Table 1 also indicates that NNCS-F and NNCS-S do better than CS on the rotated and/or shifted functions expect F_3 with the help of solutions accuracy. NNCS-F and NNCS-S not only achieve the global optimum on the function F_1, but also improve the solutions with higher accuracy to other functions. In all, according to the total number of "‡/=/†", NNCS-F and NNCS-S are significantly better than CS on 18 and 17 out of 20 functions, equal to CS on 1 and 2 out of 20 functions, and worse than CS on 1 and 1 out of 20 functions, respectively. This is because NNCS-F and NNCS-S utilize the neighbour information by learning from the nearest neighbour one in probabilistically selected dimensions, and avoid being trapped into a local optimum and a premature convergence, resulting in improving the search ability. As for the function F_3, we can observe that NNCS-F and NNCS-S do not show advantage perhaps because the effect of the nearest neighbour strategy and the probability mutation strategy are largely cancelled out by the orthogonal matrix. In addition, we can see that NNCS-F and NNCS-S are good at different functions respectively perhaps because of the similar metric to select the nearest neighbour solution.

4.2. Convergence speed study

To show how NNCS can accelerate convergence speed, Table 2 lists FES and SR obtained by CS and NNCS, where "–" means the

Table 1
Solution accuracy (mean and standard deviation) obtained by CS and NNCS at $D = 30$.

Fun	CS	NNCS-F		NNCS-S	
	$AVG_{Er} \pm SD_{Er}$		$AVG_{Er} \pm SD_{Er}$		$AVG_{Er} \pm SD_{Er}$
F_{sph}	1.46e−30±2.83e−30	‡	**1.47e−56±2.46e−56**	‡	**1.00e−53±1.46e−53**
F_{ros}	1.38e+01±1.44e+01	‡	**6.80e+00±1.01e+01**	‡	**3.80e+00±5.16e+00**
F_{ack}	1.10e−01±3.38e−01	‡	**8.24e−15±2.43e−15**	‡	**9.02e−15±2.89e−15**
F_{grw}	1.59e−15±1.12e−14	=	**0.00e+00±0.00e+00**	=	**0.00e+00±0.00e+00**
F_{ras}	2.46e+01±4.39e+00	‡	**1.72e+00±1.56e+00**	‡	**3.20e+00±1.89e+00**
F_{sch}	1.56e+03±2.39e+02	‡	**7.11e+00±3.71e+01**	‡	**2.38e+00±1.68e+01**
F_{sal}	3.72e−01±7.01e−02	‡	**2.84e−01±4.68e−02**	‡	**2.96e−01±4.50e−02**
F_{wht}	3.56e+02±7.62e+01	‡	**8.65e+01±4.66e+01**	‡	**9.72e+01±4.77e+01**
F_{pn1}	2.07e−03±1.47e−02	‡	**1.57e−32±5.53e−48**	‡	**1.57e−32±5.53e−48**
F_{pn2}	1.54e−26±8.53e−26	‡	**1.35e−32±1.11e−47**	‡	**1.35e−32±1.11e−47**
F_1	5.52e−30±1.37e−29	‡	**0.00e+00±0.00e+00**	‡	**0.00e+00±0.00e+00**
F_2	6.31e−03±6.20e−03	‡	**2.12e−03±1.28e−03**	‡	**2.89e−03±1.76e−03**
F_3	2.15e+06±5.13e+05	†	3.09e+06±8.96e+05	†	3.03e+06±7.84e+05
F_4	1.44e+03±9.05e+02	‡	**6.44e+02±6.18e+02**	‡	**6.92e+02±4.80e+02**
F_5	3.17e+03±6.76e+02	‡	**2.56e+03±6.58e+02**	‡	**2.79e+03±5.06e+02**
F_6	2.70e+01±2.56e+01	‡	**1.38e+01±1.84e+01**	=	**2.05e+01±2.44e+01**
F_7	1.06e−03±2.47e−03	‡	**6.98e−04±2.07e−03**	‡	**5.27e−04±1.68e−03**
F_8	2.093e+01±5.80e−02	‡	**2.089e+1±16.05e−02**	‡	**2.090e+01±5.72e−02**
F_9	2.94e+01±5.21e+00	‡	**1.32e+00±1.52e+00**	‡	**2.52e+00±1.59e+00**
F_{10}	1.73e+02±3.19e+01	‡	**9.81e+01±1.70e+01**	‡	**1.12e+02±1.98e+01**
‡/=/†		18/1/1		17/2/1	

algorithm can not reach to VTR. Compared to FES and SR obtained by CS, the smaller FES or the larger SR gained by NNCS are marked in boldface.

From Table 2, we can see that CS, NNCS-F and NNCS-S share the same SR performance on the functions F_{sph}, F_{grw}, F_{pn2}, F_1, and F_7. However, NNCS-F and NNCS-S show faster convergence speed than CS in terms of FES. Moreover, NNCS-F and NNCS-S outperform CS on F_{ack} and F_{pn1} in terms of both FES and SR. This means that NNCS-F and NNCS-S converge faster and more stable than CS. In addition, NNCS-F and NNCS-S can obtain FES with a certain SR performance, but CS does not. Furthermore, we can observe NNCS-F overall converges faster and more stable than NNCS-S.

To further study the convergence speed of NNCS, we select several benchmark functions to plot the evolution progress of the average and standard deviation of function error value within the maximum number of function evaluations during 50 independent runs, seen in Fig. 2. It can be observed from Fig. 2 that NNCS-F and NNCS-S converge significantly faster than CS in terms of convergence curves. Fig. 2 also reveals that NNCS-F and NNCS-S overall have smaller standard deviations of average function error. This suggests that NNCS-F and NNCS-S have steadier convergence performance.

4.3. Scalability study

In order to study the performance of NNCS affected by the problem dimensionality, a scalability study is conducted. We carry on experiments on the benchmark functions on $D = 10$ and $D = 50$ since some functions are defined up to $D = 50$ [49]. All other control parameters are unchanged from their values above mentioned except that the population size NP is 30 at $D = 10$. Solutions accuracy obtained by CS, NNCS-F and NNCS-S are listed in Table 3. Compared to the solutions accuracy obtained by CS, the higher accurate solutions gained by NNCS are marked in boldface.

According to Table 3, it suggests that the advantage of NNCS over CS is overall stable when the dimensionality of problem increases. In the case of $D = 10$, NNCS-F and NNCS-S outperform CS in terms of solution accuracy expect F_3. Also, according to the "‡/=/†" in Table 3, NNCS-F and NNCS-S are significantly better than CS on 19 out 20 functions, and worse than CS on 1 out 20 functions. When $D = 50$, NNCS-F and NNCS-S perform better than CS on all functions except F_2 and F_3 in terms of solutions accuracy, respectively. Statistically, NNCS-F and NNCS-S show better performance than CS on 18 and 17 out of 20 functions, and worse performance than CS on 2 out of 20 functions, respectively.

4.4. Influence of different solution-based similar metric

In our proposed NNCS approach, the solution-based similar metric, one of similar metric in implementation, is measured using the Euclidean distance as default. In this section, in order to investigate the influence of different distance metric on the performance of NNCS, a set of experiments are carried on. The results are listed in Table 4, where NNCS-S-SEU, NNCS-S-HAM, NNCS-S-COS, and

Table 2
FES and SR obtained by CS and NNCS at $D = 30$.

Fun	CS	NNCS-F	NNCS-S
	$AVG_{EV} \pm SD_{EV}$(SR)	$AVG_{EV} \pm SD_{EV}$(SR)	$AVG_{EV} \pm SD_{EV}$(SR)
F_{sph}	88719.6±2370.7(50)	**51270.0±1212.4**(50)	**53947.2±979.0**(50)
F_{ack}	167510.7±20034.9(45)	**79270.8±1686.8(50)**	**85087.2±2007.9(50)**
F_{grw}	125872.8±21651.2(50)	**66172.8±7208.6**(50)	**67159.2±4912.6**(50)
F_{ras}	–	**274728.0±9273.8(5)**	–
F_{sch}	–	**202227.5±18115.8(48)**	**278665.3±13350.2(38)**
F_{wht}	–	**110400.0±0.0(1)**	**186580.0±29727.6(3)**
F_{pn1}	158251.8±22648.3(49)	**54880.8±3140.1(50)**	**62006.4±3895.8(50)**
F_{pn2}	106992.0±9375.0(50)	**53370.0±1815.0**(50)	**58099.2±2472.0**(50)
F_1	93198.0±2791.5(50)	**52790.4±1265.1**(50)	**55866.0±1394.0**(50)
F_6	–	**222420.0±71700.6(2)**	–
F_7	157591.2±31064.1(50)	**138723.6±51558.3**(50)	**115134.0±24692.6**(50)
F_9	–	**271610.0±22196.7(18)**	**283920.0±7636.8(2)**

Table 3
Solution accuracy (mean and standard deviation) by CS and NNCS at $D = 10$ and $D = 50$.

Fun	$D = 10$						$D = 50$					
	CS		NNCS-F		NNCS-S		CS		NNCS-F		NNCS-S	
	$AVG_{Er} \pm SD_{Er}$		$AVG_{Er} \pm SD_{Er}$		$AVG_{Er} \pm SD_{Er}$		$AVG_{Er} \pm SD_{Er}$		$AVG_{Er} \pm SD_{Er}$		$AVG_{Er} \pm SD_{Er}$	
F_{sph}	2.97e−26±3.10e−26	‡	**1.33e−43±1.91e−43**	‡	**2.07e−39±3.17e−39**		4.72e−17±3.99e−17	‡	**2.49e−31±1.64e−31**	‡	**3.66e−30±2.38e−30**	
F_{ros}	6.85e−01±8.62e−01	‡	**1.39e−01±2.66e−01**	‡	**2.10e−01±6.05e−01**		4.21e+01±1.78e+01	‡	**3.21e+01±1.25e+01**	‡	**3.11e+01±7.41e+00**	
F_{ack}	3.92e−11±5.38e−11	‡	**3.55e−15±0.00e+00**	‡	**3.55e−15±0.00e+00**		2.87e−02±1.28e−01	‡	**1.92e−14±2.69e−15**	‡	**2.41e−14±3.07e−15**	
F_{grw}	3.62e−02±1.36e−02	‡	**8.92e−03±6.93e−03**	‡	**1.18e−02±9.59e−03**		9.05e−12±5.17e−11	‡	**0.00e+00±0.00e+00**	‡	**0.00e+00±0.00e+00**	
F_{ras}	3.15e+00±8.25e−01	‡	**1.44e−11±9.35e−11**	‡	**6.18e−02±2.22e−01**		8.69e+01±1.02e+01	‡	**3.79e+01±9.22e+00**	‡	**3.56e+01±5.76e+00**	
F_{sch}	6.79e+01±6.48e+01	‡	**3.64e−14±1.80e−13**	‡	**2.00e−09±7.70e−09**		4.93e+03±2.91e+02	‡	**6.26e+02±4.34e+02**	‡	**1.03e+03±2.75e+02**	
F_{sal}	1.06e−01±2.40e−02	‡	**9.99e−02±3.11e−11**	‡	**9.99e−02±3.47e−09**		6.99e−01±7.84e−02	‡	**4.44e−01±5.40e−02**	‡	**4.96e−01±4.93e−02**	
F_{wht}	2.28e+01±6.81e+00	‡	**2.28e+00±2.09e+00**	‡	**3.19e+00±3.64e+00**		1.39e+03±1.02e+02	‡	**5.68e+02±1.31e+02**	‡	**5.03e+02±8.42e+01**	
F_{pn1}	4.00e−18±2.24e−17	‡	**4.71e−32±1.66e−47**	‡	**4.71e−32±1.66e−47**		8.67e−03±4.23e−02	‡	**2.86e−30±3.21e−30**	‡	**2.94e−28±4.53e−28**	
F_{pn2}	3.82e−23±7.02e−23	‡	**1.35e−32±1.11e−47**	‡	**1.35e−32±1.11e−47**		2.93e−14±5.74e−14	‡	**1.53e−30±1.13e−30**	‡	**2.98e−29±2.08e−29**	
F_1	4.67e−26±4.61e−26	‡	**0.00e+00±0.00e+00**	‡	**0.00e+00±0.00e+00**		1.95e−16±1.38e−16	‡	**0.00e+00±0.00e+00**	‡	**9.04e−31±2.09e−30**	
F_2	1.04e−13±1.09e−13	‡	**3.76e−16±3.61e−16**	‡	**9.23e−16±6.96e−16**		2.41e+02±7.97e+01	†	4.52e+02±1.35e+02	†	4.57e+02±1.26e+02	
F_3	2.08e+02±1.27e+02	†	5.49e+02±4.04e+02	†	3.81e+02±2.80e+02		8.71e+06±1.35e+06	†	1.64e+07±3.28e+06	†	1.59e+07±3.01e+06	
F_4	9.59e−06±9.20e−06	‡	**1.00e−07±1.06e−07**	‡	**2.46e−07±2.29e−07**		2.83e+04±5.07e+03	‡	**2.09e+04±4.23e+03**	‡	**2.21e+04±3.92e+03**	
F_5	1.06e−04±7.72e−05	‡	**3.60e−08±2.96e−08**	‡	**5.60e−07±4.93e−07**		1.08e+04±1.41e+03	‡	**7.30e+03±9.76e+02**	‡	**7.92e+03±1.01e+03**	
F_6	1.49e+00±1.80e+00	‡	**3.84e−01±5.48e−01**	‡	**4.57e−01±9.63e−01**		6.19e+01±3.20e+01	‡	**4.36e+01±2.21e+01**	=	**5.22e+01±2.48e+01**	
F_7	5.28e−02±1.89e−02	‡	**3.38e−02±1.29e−02**	‡	**2.95e−02±1.45e−02**		1.07e−03±1.53e−03	‡	1.18e−03±3.26e−03	‡	**1.10e−04±2.31e−04**	
F_8	2.04e+01±7.11e−02	‡	**2.03e+01±6.97e−02**	‡	**2.02e+01±5.99e−02**		2.11e+01±3.11e−02	‡	2.11e+01±3.59e−02	‡	2.11e+01±3.81e−02	
F_9	2.78e+00±1.03e+00	‡	**9.69e−13±6.06e−12**	‡	**5.60e−04±3.79e−03**		1.25e+02±1.24e+01	‡	**3.47e+01±8.05e+00**	‡	**3.49e+01±5.20e+00**	
F_{10}	1.96e+01±5.96e+00	‡	**1.45e+01±4.43e+00**	‡	**1.53e+01±5.26e+00**		4.00e+02±4.75e+01	‡	**2.33e+02±2.92e+01**	‡	**2.50e+02±3.14e+01**	
‡/=/†		19/0/1		19/0/1				18/0/2		17/1/2		

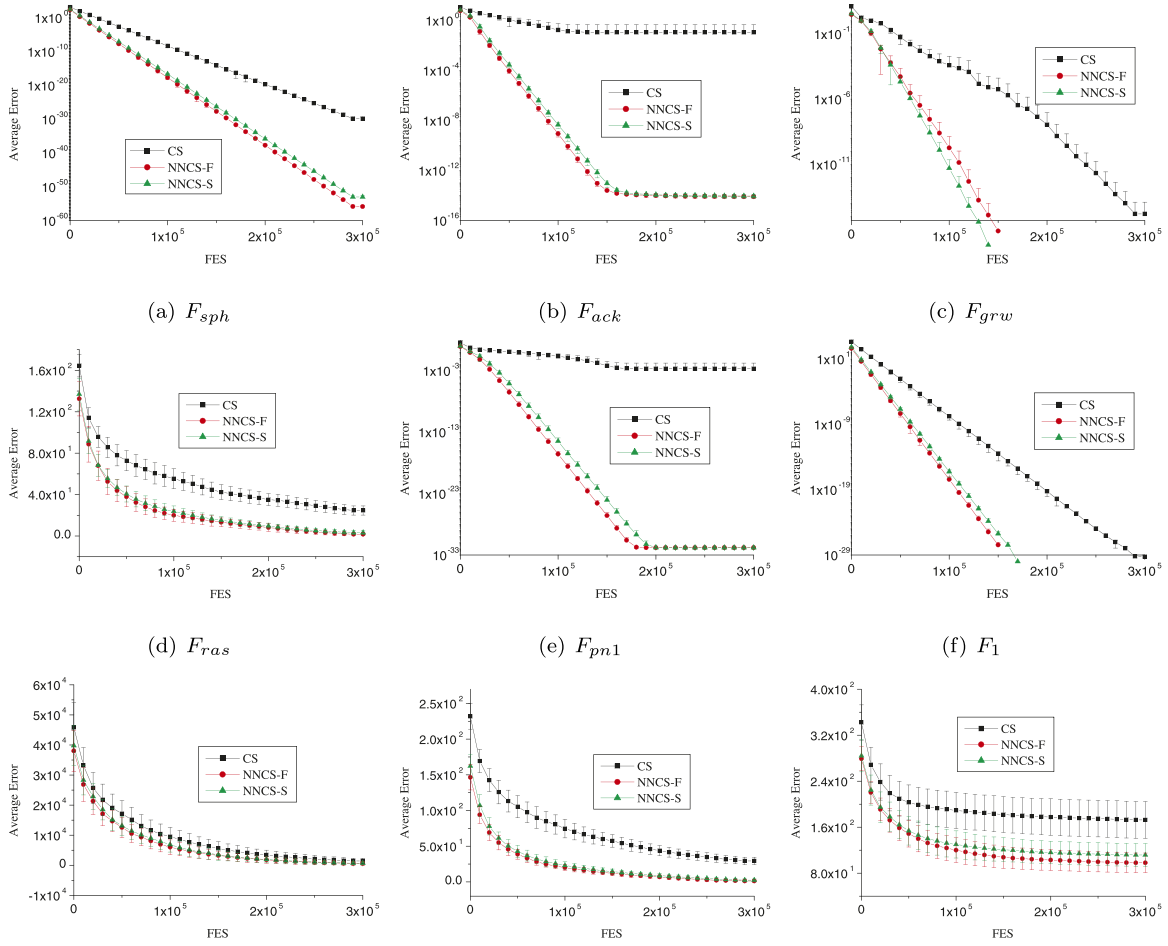


Fig. 2. Convergence graphs of CS and NNCS for selected functions.

Table 4
Solution accuracy (mean and standard deviation) obtained by NNCS using different distance metric at $D = 30$.

*Fun	NNCS-S-SEU	NNCS-S-HAM	NNCS-S-MAN	NNCS-S-COS	NNCS-S
	$AVG_{Er} \pm SD_{Er}$	$AVG_{Er} \pm SD_{Er}$	$AVG_{Er} \pm SD_{Er}$	$AVG_{Er} \pm SD_{Er}$	$AVG_{Er} \pm SD_{Er}$
F_{sph}	7.18e−54 ± 8.11e−54	3.37e−51 ± 9.30e−51	**5.33e−54 ± 6.18e−54**	4.10e−52 ± 7.57e−52	1.00e−53 ± 1.46e−53
F_{ros}	4.69e+00 ± 5.83e+00	5.68e+00 ± 6.12e+00	5.70e+00 ± 1.06e+01	4.78e+00 ± 9.53e+00	**3.80e+00 ± 5.16e+00**
F_{ack}	**8.31e−15 ± 2.34e−15**	8.46e−15 ± 2.58e−15	8.38e−15 ± 2.56e−15	8.74e−15 ± 2.89e−15	9.02e−15 ± 2.89e−15
F_{grw}	0.00e+00 ± 0.00e+00	0.00e+00 ± 0.00e+00	0.00e+00 ± 0.00e+00	0.00e+00 ± 0.00e+00	0.00e+00 ± 0.00e+00
F_{ras}	2.51e+00 ± 1.91e+00	7.55e+00 ± 3.24e+00	**2.41e+00 ± 1.77e+00**	3.85e+00 ± 2.03e+00	3.20e+00 ± 1.89e+00
F_{sch}	1.32e−04 ± 7.78e−04	1.78e+01 ± 6.90e+01	2.37e+00 ± 1.67e+01	**1.82e−05 ± 9.06e−05**	2.38e+00 ± 1.68e+01
F_{sal}	3.08e−01 ± 5.66e−02	**2.84e−01 ± 4.22e−02**	3.06e−01 ± 5.12e−02	2.98e−01 ± 4.73e−02	2.96e−01 ± 4.50e−02
F_{wht}	9.69e+01 ± 4.56e+01	1.33e+02 ± 4.52e+01	**9.25e+01 ± 3.68e+01**	1.02e+02 ± 5.01e+01	9.72e+01 ± 4.77e+01
F_{pn1}	1.57e−32 ± 5.53e−48	1.57e−32 ± 5.53e−48	1.57e−32 ± 5.53e−48	1.57e−32 ± 5.53e−48	1.57e−32 ± 5.53e−48
F_{pn2}	1.35e−32 ± 1.11e−47	1.35e−32 ± 1.11e−47	1.35e−32 ± 1.74e−34	1.35e−32 ± 1.11e−47	1.35e−32 ± 1.11e−47
F_1	0.00e+00 ± 0.00e+00	0.00e+00 ± 0.00e+00	0.00e+00 ± 0.00e+00	0.00e+00 ± 0.00e+00	0.00e+00 ± 0.00e+00
F_2	2.63e−03 ± 1.76e−03	4.68e−03 ± 3.40e−03	**2.45e−03 ± 1.66e−03**	2.58e−03 ± 1.37e−03	2.89e−03 ± 1.76e−03
F_3	2.97e+06 ± 8.63e+05	3.09e+06 ± 9.33e+05	**2.92e+06 ± 9.57e+05**	3.15e+06 ± 9.45e+05	3.03e+06 ± 7.84e+05
F_4	**6.35e+02 ± 3.88e+02**	5.81e+02 ± 4.12e+02	6.67e+02 ± 3.72e+02	9.18e+02 ± 6.56e+02	6.92e+02 ± 4.80e+02
F_5	2.73e+03 ± 6.54e+02	**2.35e+03 ± 5.57e+02**	2.75e+03 ± 6.86e+02	2.62e+03 ± 5.67e+02	2.79e+03 ± 5.06e+02
F_6	1.63e+01 ± 1.90e+01	**1.21e+01 ± 1.46e+01**	2.10e+01 ± 2.34e+01	1.96e+01 ± 1.96e+01	2.05e+01 ± 2.44e+01
F_7	6.73e−04 ± 2.02e−03	1.06e−03 ± 2.41e−03	**2.97e−04 ± 1.46e−03**	3.78e−04 ± 1.50e−03	5.27e−04 ± 1.68e−03
F_8	**2.089e+01 ± 5.50e−02**	**2.089e+01 ± 5.08e−02**	**2.089e+01 ± 5.67e−02**	2.090e+01 ± 4.83e−02	2.090e+01 ± 5.72e−02
F_9	2.24e+00 ± 1.45e+00	6.41e+00 ± 2.78e+00	**2.20e+00 ± 1.81e+00**	2.60e+00 ± 1.78e+00	2.52e+00 ± 1.59e+00
F_{10}	1.12e+02 ± 2.13e+01	**1.04e+02 ± 1.57e+01**	1.11e+02 ± 2.38e+01	1.08e+02 ± 1.92e+01	1.12e+02 ± 1.98e+01

Table 5
Average ranking of five NNCS using different distance metric by the Friedman test for 20 functions at $D = 30$.

Algorithm	NNCS-S-SEU	NNCS-S-HAM	NNCS-S-MAN	NNCS-S-COS	NNCS-S
Ranking	2.88	3.08	**2.82**	3.16	*3.06*

The significance of bold is to show the smallest average ranking.

Table 6
Solution accuracy (mean) obtained by NNCS-F using different mutation probability at $D = 30$.

Fun	NNCS-0.1	NNCS-0.2	NNCS-0.25	NNCS-0.3	NNCS-0.4	NNCS-0.5	NNCS-0.6	NNCS-0.7	NNCS-0.8	NNCS-0.9
F_{sph}	4.69E−47	2.98E−55	1.47E−56	2.61E−57	**1.67E−57**	6.53E−57	9.74E−56	1.90E−54	3.54E−53	5.72E−52
F_{ros}	8.93E+00	7.05E+00	6.80E+00	5.31E+00	2.49E+00	**2.46E+00**	4.22E+00	4.30E+00	3.45E+00	4.59E+00
F_{ack}	9.24E−15	8.03E−15	8.24E−15	8.17E−15	7.11E−15	7.18E−15	7.39E−15	7.25E−15	7.25E−15	**7.03E−15**
F_{grw}	0.00E+00	0.00E+00	0.00E+00	0.00E+00	0.00E+00	1.48E−04	0.00E+00	0.00E+00	0.00E+00	0.00E+00
F_{ras}	**8.13E−02**	1.03E+00	1.72E+00	3.34E+00	4.80E+00	6.89E+00	9.36E+00	1.18E+01	1.28E+01	1.43E+01
F_{sch}	**5.46E−13**	2.37E+00	7.11E+00	2.37E+00	3.55E+01	7.00E+01	1.33E+02	2.44E+02	3.35E+02	3.62E+02
F_{sal}	3.36E−01	2.92E−01	2.84E−01	2.86E−01	2.48E−01	2.40E−01	**2.16E−01**	2.28E−01	2.22E−01	2.20E−01
F_{wht}	**3.37E+01**	7.69E+01	8.65E+01	1.12E+02	1.40E+02	1.60E+02	2.00E+02	2.01E+02	2.16E+02	2.21E+02
F_{pn1}	1.57E−32	1.57E−32	1.57E−32	1.57E−32	1.57E−32	1.57E−32	1.57E−32	1.57E−32	1.57E−32	1.57E−32
F_{pn2}	1.35E−32	1.35E−32	1.35E−32	1.35E−32	1.35E−32	1.35E−32	1.35E−32	1.35E−32	1.35E−32	1.35E−32
F_1	0.00E+00	0.00E+00	0.00E+00	0.00E+00	0.00E+00	0.00E+00	0.00E+00	0.00E+00	0.00E+00	0.00E+00
F_2	7.32E−03	2.96E−03	**2.12E−03**	2.74E−03	3.19E−03	3.70E−03	4.43E−03	7.25E−03	6.83E−03	9.93E−03
F_3	**2.88E+06**	3.10E+06	3.09E+06	3.08E+06	3.21E+06	3.21E+06	3.17E+06	3.10E+06	3.29E+06	3.42E+06
F_4	1.08E+03	6.65E+02	6.44E+02	5.59E+02	4.23E+02	3.84E+02	4.73E+02	3.48E+02	4.55E+02	**3.42E+02**
F_5	2.86E+03	2.70E+03	2.56E+03	2.42E+03	2.29E+03	2.22E+03	2.09E+03	2.14E+03	2.09E+03	**1.74E+03**
F_6	1.63E+01	1.93E+01	1.38E+01	1.57E+01	1.27E+01	1.11E+01	**9.30E+00**	1.16E+01	1.93E+01	1.14E+01
F_7	**5.75E−04**	7.10E−04	6.98E−04	1.10E−03	1.48E−03	3.04E−03	3.48E−03	2.38E−03	1.97E−03	2.40E−03
F_8	**2.08E+01**	2.09E+01	2.09E+01	2.09E+01	2.09E+01	2.09E+01	2.09E+01	2.09E+01	2.09E+01	2.09E+01
F_9	**4.00E−02**	7.24E−01	1.32E+00	2.78E+00	4.04E+00	6.98E+00	8.28E+00	1.17E+01	1.26E+01	1.61E+01
F_{10}	1.16E+02	1.04E+02	9.81E+01	9.52E+01	9.38E+01	8.84E+01	**8.76E+01**	8.92E+01	8.97E+01	8.85E+01

Table 7
Solution accuracy (mean) obtained by NNCS-S using different mutation probability at $D = 30$.

Fun	NNCS-0.1	NNCS-0.2	NNCS-0.25	NNCS-0.3	NNCS-0.4	NNCS-0.5	NNCS-0.6	NNCS-0.7	NNCS-0.8	NNCS-0.9
F_{sph}	1.89E−44	3.79E−52	1.00E−53	1.25E−54	**2.06E−55**	4.68E−55	4.82E−54	3.45E−53	2.92E−52	4.91E−51
F_{ros}	7.09E+00	4.40E+00	3.80E+00	5.93E+00	3.48E+00	5.85E+00	**2.54E+00**	4.00E+00	3.67E+00	3.97E+00
F_{ack}	1.00E−14	8.10E−15	9.02E−15	7.89E−15	7.89E−15	7.96E−15	7.82E−15	7.39E−15	7.39E−15	**7.32E−15**
F_{grw}	0.00E+00	0.00E+00	0.00E+00	0.00E+00	0.00E+00	0.00E+00	0.00E+00	0.00E+00	0.00E+00	0.00E+00
F_{ras}	**2.66E+00**	2.67E+00	3.20E+00	3.97E+00	4.97E+00	7.44E+00	9.70E+00	1.13E+01	1.31E+01	1.31E+01
F_{sch}	1.29E+01	2.37E+00	2.38E+00	**3.88E−02**	4.75E+00	2.70E+01	8.93E+01	7.64E+01	1.74E+02	2.47E+02
F_{sal}	3.44E−01	3.08E−01	2.96E−01	2.92E−01	2.82E−01	2.64E−01	2.60E−01	**2.42E−01**	2.60E−01	2.48E−01
F_{wht}	**6.58E+01**	8.29E+01	9.72E+01	1.02E+02	1.26E+02	1.40E+02	1.53E+02	1.64E+02	1.92E+02	1.92E+02
F_{pn1}	1.57E−32	1.57E−32	1.57E−32	1.57E−32	1.57E−32	1.57E−32	1.57E−32	1.57E−32	1.57E−32	1.57E−32
F_{pn2}	1.35E−32	1.35E−32	1.35E−32	1.35E−32	1.35E−32	1.35E−32	1.35E−32	1.35E−32	1.35E−32	1.35E−32
F_1	0.00E+00	0.00E+00	0.00E+00	0.00E+00	0.00E+00	0.00E+00	0.00E+00	0.00E+00	0.00E+00	0.00E+00
F_2	5.88E−03	3.20E−03	2.89E−03	**2.49E−03**	3.04E−03	3.84E−03	4.02E−03	4.21E−03	7.62E−03	7.26E−03
F_3	**2.67E+06**	2.89E+06	3.03E+06	3.03E+06	3.28E+06	2.97E+06	3.03E+06	3.20E+06	3.26E+06	3.33E+06
F_4	1.25E+03	7.81E+02	6.92E+02	6.02E+02	4.99E+02	5.27E+02	4.61E+02	**4.14E+02**	4.35E+02	5.06E+02
F_5	2.77E+03	2.82E+03	2.79E+03	2.57E+03	2.46E+03	2.30E+03	2.37E+03	2.25E+03	**2.05E+03**	2.17E+03
F_6	2.66E+01	1.87E+01	2.05E+01	2.11E+01	2.45E+01	1.94E+01	**1.56E+01**	2.06E+01	2.41E+01	1.68E+01
F_7	6.22E−04	3.91E−04	5.27E−04	**8.62E−05**	7.16E−04	5.39E−04	5.50E−04	5.42E−04	7.59E−04	2.11E−03
F_8	**2.07E+01**	2.08E+01	2.09E+01	2.09E+01	2.09E+01	2.09E+01	2.09E+01	2.09E+01	2.09E+01	2.09E+01
F_9	3.09E+00	**1.85E+00**	2.52E+00	3.15E+00	4.99E+00	6.67E+00	8.25E+00	9.79E+00	1.17E+01	1.30E+01
F_{10}	1.29E+02	1.15E+02	1.12E+02	1.09E+02	1.03E+02	9.92E+01	9.83E+01	9.76E+01	**9.69E+01**	9.87E+01

NNCS-S-MAN mean the standardized Euclidean distance, Hamming distance, Cosine distance, and Manhattan distance are used in NNCS, respectively. The highest accurate solution to each function is marked in boldface. In addition, Table 5 reports the ranking analysis between NNCS-S and NNCS-S with other distance metrics for the 20 functions based on the Friedman test similar done in [52].

From Table 4, in terms of solution accuracy, NNCS-S-SEU and NNCS-S-COS achieve the significantly higher accurate solutions to F_{sch}. For other functions, the five algorithms exhibit the equivalent accuracy. For example, we can see that the five NNCS-S algorithms achieve the global optimum on two functions F_{grw} and F_1. However, according to the average ranking in Table 5, NNCS-S is inferior to NNCS-S-SEU and NNCS-S-MAN, and followed by NNCS-S-HAM and NNCS-S-COS. This suggests that the performance of NNCS-S is influenced by different solution-based distance metrics. It also shows that the Euclidean distance is not an optimal but a reasonable choice compared with the other four distance metrics.

4.5. Influence of the mutation probability

In our proposed NNCS algorithm, the mutation probability p is used to control that the solutions learn from the nearest neighbour ones in probabilistically selected dimensions. In this section, we perform additional experiments to study the performance of NNCS influenced by the mutation probability p. The mutation probability p is set from 0.1 to 0.9 with a step size of 0.1. The results are shown in Tables 6 and 7, where NNCS-n means the mutation probability $p = n$ used in NNCS. The highest accurate solution to each function is marked in boldface.

From Table 6, we can see that the mutation probability p overall has slightly significant effect on the performance of NNCS-F for some functions. For example, NNCS-F obtains the better performance with a lower mutation probability p on F_{ras}, F_{grw}, F_{sch}, F_{wht}, F_2, F_3, F_7, F_8, and F_9. On the contrary, NNCS-F does well on F_{sph}, F_{ros}, F_{ack}, F_{sal}, F_4, F_5, F_6, and F_{10} in the case of a higher mutation probability. However, there are still some functions which are indifferent to the mutation probability p. For instance, NNCS-F almost achieves the similar performance on F_{pn1}, F_{pn2}, and F_1. Additionally, from Table 7, we almost can draw the same conclusion on NNCS-S.

Nevertheless, Table 8 reports the results of Friedman test for these algorithms with different mutation probabilities. According to the average ranking, it is observed that a trade-off interval of the mutation probability is from 0.2 to 0.6. It suggests that the mutation probability set 0.25 is suitable for the given 20 benchmark functions.

Table 8
Average ranking of 10 algorithms by the Friedman test for 20 functions at $D = 30$.

Algorithm	NNCS-0.1	NNCS-0.2	NNCS-0.25	NNCS-0.3	NNCS-0.4	NNCS-0.5	NNCS-0.6	NNCS-0.7	NNCS-0.8	NNCS-0.9
NNCS-F ranking	5.75	5.40	5.05	5.05	**4.75**	5.25	5.30	5.72	6.28	6.45
NNCS-S ranking	6.35	5.10	5.15	**4.83**	5.38	5.35	5.00	5.42	5.97	6.45

The significance of bold is to show the smallest average ranking.

Table 9
Solution accuracy (mean and standard deviation) obtained by various CS algorithms at $D = 30$.

Fun	ICS		CSPSO		OLCS		NNCS-F	NNCS-S
F_{sph}	9.43e−49	‡(‡)	2.83e−44	‡(‡)	**3.20e−128**	†(†)	1.47e−56	1.00e−53
F_{ros}	9.62e+00	‡(‡)	**4.87e−01**	†(†)	2.49e+00	†(=)	6.80e+00	3.80e+00
F_{ack}	1.02e−14	‡(‡)	5.95e−14	‡(‡)	**2.66e−15**	†(†)	8.24e−15	9.02e−15
F_{grw}	**0.00e+00**	‡(=)	8.02e−03	‡(‡)	**0.00e+00**	=(=)	**0.00e+00**	**0.00e+00**
F_{ras}	1.80e+01	‡(‡)	2.84e+01	‡(‡)	**0.00e+00**	†(†)	1.72e+00	3.20e+00
F_{sch}	7.41e+02	‡(‡)	4.48e+03	‡(‡)	2.13e+03	‡(‡)	7.11e+00	**2.38e+00**
F_{sal}	2.12e−01	‡(†)	4.26e−01	‡(‡)	**5.15e−07**	†(†)	2.84e−01	2.96e−01
F_{wht}	2.66e+02	‡(‡)	4.83e+02	‡(‡)	3.15e+02	‡(‡)	**8.65e+01**	9.72e+01
F_{pn1}	**1.57e−32**	=(=)	5.81e−02	‡(‡)	5.01e−30	‡(‡)	**1.57e−32**	**1.57e−32**
F_{pn2}	**1.35e−32**	=(=)	1.54e−03	‡(‡)	4.46e−29	‡(‡)	**1.35e−32**	**1.35e−32**
F_1	**0.00e+00**	=(=)	2.71e−28	‡(‡)	2.41e−26	‡(‡)	**0.00e+00**	**0.00e+00**
F_2	1.43e−03	†(†)	**1.98e−11**	†(†)	5.70e−02	‡(‡)	2.12e−03	2.89e−03
F_3	**3.55e+05**	†(†)	7.81e+05	†(†)	2.57e+06	†(†)	3.09e+06	3.03e+06
F_4	3.50e+02	†(†)	**8.02e+01**	†(†)	2.37e+03	‡(‡)	6.44e+02	6.92e+02
F_5	**1.67e+03**	†(†)	3.20e+03	‡(‡)	2.44e+03	=(†)	2.56e+03	2.79e+03
F_6	1.27e+01	=(=)	**5.10e+00**	†(†)	2.45e+01	‡(‡)	1.38e+01	2.05e+01
F_7	2.64e−03	‡(‡)	1.57e−02	‡(‡)	**4.72e−04**	‡(‡)	6.98e−04	5.27e−04
F_8	2.094e+01	‡(‡)	2.094e+01	‡(‡)	2.094e+01	‡(‡)	**2.089e+01**	2.090e+01
F_9	1.48e+01	‡(‡)	1.54e+02	‡(‡)	3.54e+01	‡(‡)	**1.32e+00**	2.52e+00
F_{10}	**7.91e+01**	†(†)	2.61e+02	‡(‡)	1.54e+02	‡(‡)	9.81e+01	1.12e+02
‡/=/†	9/5/6(8/6/6)		15/0/5(15/0/5)		12/2/6(12/2/6)			

Table 10
Average ranking of five algorithms by the Friedman test for 20 functions at $D = 30$.

Algorithm	ICS	CSPSO	OLCS	NNCS-S	NNCS-F
Ranking	2.73	3.95	3.08	2.83	**2.42**

The significance of bold is to show the smallest average ranking.

4.6. Comparison with other improved CS versions

In this section, to show the competitiveness of NNCS, the proposed algorithm is compared with three improved CS variants, namely, ICS [24], CSPSO [25], and OLCS [27,28], although there are many variants CS. The average of the function value error is given in Table 9, and the highest accurate solution to each function is marked in boldface. Also, we perform the Wilcoxon signed-rank test between each improved algorithm and NNCS, and the results are denoted as "a(b)" where "a" and "b" which are the significant symbols (‡ = †). The significant symbol out of the bracket means the result between the compared algorithm and NNCS-F, and the significant symbol in the bracket is the result between the compared algorithm and NNCS-S. In addition, the ranking analysis are performed between NNCS and three algorithms for the 20 functions by using Friedman test, and the results are reported in Table 10.

In terms of solution accuracy in Table 9, each algorithm is good at parts of functions. In other words, some may provide better solutions and faster convergence for particular optimization problems than others. For example, in terms of solution accuracy, ICS can obtain the global optimum on F_{grw} and F_1, and brings the solutions with higher accuracy F_{pn1}, F_{pn2}, F_3, F_5, and F_{10}. CSPSO does best in optimizing F_{ros}, F_2, F_4, and F_6. OLCS seems to be good at some multimodal functions, where OLCS achieves the global optimum on F_{grw} and F_{ras}, and yields higher solution accuracy on F_{sph}, F_{ack}, F_{sal}, and F_7. NNCS-F and NNCS-S yield the global optimum on F_{grw} and F_1, and bring the solutions with higher accuracy on F_{sch}, F_{wht}, F_{pn1}, F_{pn2}, F_8, and F_9. Moreover, according to the "‡/=/†" in Table 9, NNCS-F is superior to ICS, CSPSO, and OLCS on 9, 15 and 12 out of 20 functions, equivalent to ICS, CSPSO and OLCS on 5, 0 and 2 out of 20 ones, and inferior to ICS, CSPSO, and OLCS on 6, 5 and 6 out of 20 ones, respectively. Similarly, NNCS-S outperforms ICS, CSPSO, and OLCS on 8, 15 and 12 out of 20 functions, equals to ICS, CSPSO and OLCS on 6, 0 and 2 out of 20 ones, and loses ICS, CSPSO, and OLCS on 6, 5 and 6 out of 20 ones, respectively. Nevertheless, in terms of the average ranking of five algorithms, we can see from Table 10 that NNCS-F achieves the smallest average ranking, followed by ICS, NNCS-S, OLCS, and CSPSO. This suggests that NNCS-F is overall best.

4.7. Comparison with other algorithms

In this section, NNCS is compared with two state-of-the-art algorithms, namely, DEahcSPX [50] and CMA-ES [48] that employed a self-adaptation of arbitrary mutation strategy. Except that the maximum number of function evaluations (MaxFEs) is $10{,}000 \times D$, CMA-ES whose parameters and source code are similar done in [48] is performed for the 20 benchmark functions, and the compared results are listed in Table 11. The results obtained by DEahcSPX are taken from the literature [50]. We also mark the highest accurate solution to each function in boldface. Additionally, Table 12 reports average ranking of four algorithms for 20 functions at $D = 30$ by using Friedman test.

Seen from Table 11, DEahcSPX, CMA-ES, NNCS-S and NNCS-F exhibit different advantage over different functions. Compared with other algorithms, DEahcSPX yields 5 out of 20 functions in terms of accuracy. For F_{ros}, F_2, F_3, F_5, F_6, and F_{10}, CMA-ES achieves the solutions with higher accuracy according to average error. However, NNCS-S and NNCS-F obtain the higher accurate solutions to

Table 11
Solution accuracy (mean and standard deviation) obtained by DEahcSPX, CMA-ES and NNCS at $D = 30$.

Fun	DEahcSPX	CMA-ES	NNCS-S	NNCS-F
	$AVG_{Er} \pm SD_{Er}$	$AVG_{Er} \pm SD_{Er}$	$AVG_{Er} \pm SD_{Er}$	$AVG_{Er} \pm SD_{Er}$
F_{sph}	1.75e−31±4.99e−31	6.09e−29±1.42e−29	1.00e−53±1.46e−53	**1.47e−56±2.46e−56**
F_{ros}	4.52e+00±1.55e+01	**3.19e−01±1.09e+00**	3.80e+00±5.16e+00	6.80e+00±1.01e+01
F_{ack}	**2.66e−15±0.00e+00**	1.47e−14±3.59e−15	9.02e−15±2.89e−15	8.24e−15±2.43e−15
F_{grw}	2.07e−03±5.89e−03	1.33e−17±3.64e−17	**0.00e+00±0.00e+00**	**0.00e+00±0.00e+00**
F_{ras}	2.14e+01±1.23e+01	5.06e+01±1.12e+01	3.20e+00±1.89e+00	**1.72e+00±1.56e+00**
F_{sch}	4.70e+02±2.96e+02	1.25e+04±2.29e−12	**2.38e+00±1.68e+01**	7.11e+00±3.71e+01
F_{sal}	**1.80e−01±4.08e−02**	8.92e−01±1.38e−01	2.96e−01±4.50e−02	2.84e−01±4.68e−02
F_{wht}	3.06e+02±1.10e+02	4.15e+02±6.13e+01	9.72e+01±4.77e+01	**8.65e+01±4.66e+01**
F_{pn1}	2.07e−02±8.46e−02	2.49e−02±4.94e−02	**1.57e−32±5.53e−48**	**1.57e−32±5.53e−48**
F_{pn2}	1.71e−31±5.35e−31	6.59e−04±2.64e−03	**1.35e−32±1.11e−47**	**1.35e−32±1.11e−47**
F_1	**0.00e+00±0.00e+00**	6.08e−26±4.67e−26	**0.00e+00±0.00e+00**	**0.00e+00±0.00e+00**
F_2	6.52e−05±4.84e−05	**2.35e−25±7.69e−26**	2.89e−03±1.76e−03	2.12e−03±1.28e−03
F_3	1.29e+06±9.22e+05	**1.24e−21±4.01e−22**	3.03e+06±7.84e+05	3.09e+06±8.96e+05
F_4	**4.62e+00±8.78e+00**	8.83e+03±6.91e+03	6.92e+02±4.80e+02	6.44e+02±6.18e+02
F_5	9.00e+02±4.79e+02	**1.21e−10±2.93e−11**	2.79e+03±5.06e+02	2.56e+03±6.58e+02
F_6	3.84e+00±3.75e+00	**7.97e−01±1.61e+00**	2.05e+01±2.44e+01	1.38e+01±1.84e+01
F_7	7.39e−03±6.32e−03	2.32e−03±4.81e−03	**5.27e−04±1.68e−03**	6.98e−04±2.07e−03
F_8	**2.09e+01±1.12e−01**	2.10e+01±6.97e−01	**2.09e+01±5.72e−02**	**2.09e+01±6.05e−02**
F_9	2.04e+01±8.19e+00	4.99e+01±1.39e+01	2.52e+00±1.59e+00	**1.32e+00±1.52e+00**
F_{10}	5.27e+01±4.84e+01	**4.69e+01±1.15e+01**	1.12e+02±1.98e+01	9.81e+01±1.70e+01

Table 12
Average ranking of four algorithms by the Friedman test for 20 functions at $D = 30$.

Algorithm	DEahcSPX	CMA-ES	NNCS-S	NNCS-F
Ranking	2.45	3.00	2.42	**2.13**

The significance of bold is to show the smallest average ranking.

most of multimodal functions and parts of shifted and/or rotated functions. Nevertheless, with respect to the mean ranking of four algorithms by Friedman test, Table 12 shows that NNCS-F is overall best, and NNCS-S is second best, following by DEahcSPX and CMA-ES. This suggests that NNCS-F and NNCS-S overall outperforms CMA-ES and DEahcSPX for these 20 functions.

5. Conclusion and future work

In this paper, we proposed NNCS, an improved CS algorithm with the nearest neighbour strategy and the probabilistic mutation strategy. The nearest neighbour strategy helped the proposed algorithm to generate new solutions learning from their nearest neighbour solutions instead of the best solution obtained so far. In the nearest neighbour strategy, we used a solution-based and a fitness-based similar metrics to select the nearest neighbour solutions. The probabilistic mutation strategy was used to control the solutions learn from the nearest neighbour solutions in partial dimensions instead of all of them in conventional CS. Moreover, our experiments indicated the improvement in effectiveness and efficiency of the nearest neighbour strategy and the probability mutation strategy. The results also revealed that the advantage of NNCS over CS was overall steady as the dimension of problem increases. Additionally, compared with NNCS-S, NNCS-F is a recommended algorithm in terms of solution accuracy, convergence speed, and the convenience of similar metric.

The proposed algorithm has the following unique characteristics: (i) in LFRW, it utilizes the non-wheel topology instead of wheel topology where the best solution influences all other solutions; (ii) it makes the best solution participate in searching according to Eq. (14); (iii) it uses a solution-based and a fitness-based similar metrics to select the nearest neighbour solutions, respectively; (iv) it overall brings solutions with higher accuracy and faster convergence speed; (v) it can deal with high-dimensional optimization problems in terms of scalability study.

There are several interesting directions for future work. Firstly, it is interesting to test other strategy to select nearest neighbour solutions for other solutions learning. Secondly, we plan to investigate the self-adaptive version of the mutation probability. Last but not least, we also plan to apply NNCS to more benchmark functions including some real-world optimization problems for further examinations.

Acknowledgements

The authors thank the anonymous reviewers for their very helpful and constructive comments and suggestions. This work was supported by the NSFC Joint Fund with Guangdong of China under Key Project U1201258, the National Natural Science Foundation of China under Grant No. 61573219, the Shandong Natural Science Funds for Distinguished Young Scholar under Grant No. JQ201316, the Fostering Project of Dominant Discipline and Talent Team of Shandong Province Higher Education Institutions, and the Natural Science Foundation of Fujian Province of China under Grant No. 2016J01280, and Major projects of regional development of Fujian Province of China under Grant No. 2015N3011.

Appendix A. Appendixes

The 20 benchmark functions were as follows.
F_{sph}: Sphere's Function.
F_{ros}: Rosenbrock's Function.
F_{ack}: Ackley's Function.
F_{grw}: Griewank's Function.
F_{ras}: Rastrigin's Function.
F_{sch}: Generalized Schwefel's Problem 2.26.
F_{sal}: Salomon's Function.
F_{wht}: Whitely's Function.
F_{pn1}: Generalized Penalized Function 1.
F_{pn2}: Generalized Penalized Function 2.
F_1: Shifted Sphere Function.
F_2: Shifted Schwefel's Problem 1.2.
F_3: Shifted Rotated High Conditioned Elliptic Function.
F_4: Shifted Schwefel's Problem 1.2 with Noise in Fitness.

F_5: Schwfel's Problem 2.6 with global Optimum on Bounds.
F_6: Shifted Rosenbrock's Function.
F_7: Shifted Rotated Griewank's Function without Bounds.
F_8: Shifted Rotated Ackley's Function with Global Optimum on Bounds.
F_9: Shifted Rastrigin' Function.
F_{10}: Shifted Rotated Rastrigin's Function.

Appendix B.

```
Demo implementation
function [bestnest,fmin] = NNCSDemo
  % the following code is implemented according to the code provided in
  [20]
  NP = 30; % the population size
  D = 30; % the dimension size
  maxFES = 10,000*D;
  pa = 0.25;% a fraction probability
  p = 0.25; % a probability to control each solution learning for the nearest
  neigbour one, seen as Eq. (15)
  beta = 3/2;
  sigma = (gamma(1+beta)*sin(pi*beta/2)/(gamma((1+beta)/2)*beta*2^((beta-
  1)/2)))^(1/beta);
  smType = 0; %0: fitness-based similar metric; 1: solution-based similar
  metric
  lu(1,1:D) = −100*ones(1,D); %lower search space
  lu(2,1:D) = 100*ones(1,D); % upper search space
  nest = repmat(lu(1,:), NP, 1) + rand(NP, D).* (repmat(lu(2,:) − lu(1,:), NP,
  1));
  fits = fobj(nest);
  FES = NP;
  [fmin,min_i] = min(fits);
  bestnest = nest(min_i,:);
  new_nest = zeros(NP,D);
  while FES < maxFES
    pK = rand(size(nest))<p;
    if smType = = 1
      Dis = squareform(pdist(nest,'euclidean'));
    end
    for i = 1:NP
      if smType = = 1
        Dis(i,i) = inf;
        [ nearestID] = min(Dis(i,:));
        ,
      else
        Dis = fits-fits(i);
        Dis(i) = inf;
        [ nearestID] = min(Dis);
        ,
      end
      u = randn(1,D)*sigma;
      v = randn(1,D);
      step = u./abs(v).^(1/beta);
      stepsize = rand.*step.*(nest(i,:)-nest(nearestID,:));
      new_nest(i,:) = nest(i,:)+stepsize.*randn(1,D).*pK(i,:);
    end
    new_nest = simplebounds(new_nest,lu(1,:),lu(2,:));
    new_fits = fobj(new_nest);
    FES = FES+NP;
    iBetter = new_fits<fits;
    fits(iBetter) = new_fits(iBetter);
    nest(iBetter,:) = new_nest(iBetter,:);
    %biased random walk component
    paK = rand(size(nest))>pa;
    stepsize = rand*(nest(randperm(NP),:)-nest(randperm(NP),:));
    new_nest = nest+stepsize.*paK;
    new_nest = simplebounds(new_nest,lu(1,:),lu(2,:));
    new_fits = fobj(new_nest);
    FES = FES+NP;
    iBetter = new_fits<fits;
    fits(iBetter) = new_fits(iBetter);
    nest(iBetter,:) = new_nest(iBetter,:);
    [fmin,min_i] = min(fits);
    bestnest = nest(min_i,:);
  end
function z = fobj(x)    z = sum(x.^2, 2);
function s = simplebounds(s,Lbb,Ubb)
  % the following code comes from the implementation code of the
  literature [45]
  NP = size(s,1);
  Lb = repmat(Lbb, NP, 1);
  Ub = repmat(Ubb, NP, 1);
  ns_tmp = s;
  % Check the lower bound
  I = ns_tmp<Lb;
  ns_tmp(I) = 2.*Lb(I)-ns_tmp(I);
  I_ = ns_tmp(I)>Ub(I);
  ns_tmp(I(I_)) = Ub(I(I_));
  % Check the upper bounds
  J = ns_tmp>Ub;
  ns_tmp(J) = 2.*Ub(J)-ns_tmp(J);
  J_ = ns_tmp(J)<Lb(J);
  ns_tmp(J(J_)) = Lb(J(J_));
  s = ns_tmp;
```

References

[1] R.-C. David, R.-E. Precup, E.M. Petriu, M.-B. Rădac, S. Preitl, Gravitational search algorithm-based design of fuzzy control systems with a reduced parametric sensitivity, Inf. Sci. 247 (2013) 154–173.
[2] A.-C. Zăvoianu, G. Bramerdorfer, E. Lughofer, S. Silber, W. Amrhein, E.P. Klement, Hybridization of multi-objective evolutionary algorithms and artificial neural networks for optimizing the performance of electrical drives, Eng. Appl. Artif. Intell. 26 (8) (2013) 1781–1794.
[3] N. El-Hefnawy, Solving bi-level problems using modified particle swarm optimization algorithm, Int. J. Artif. Intell. 12 (2) (2014) 88–101.
[4] J. Holland, Adaptation in Natural and Artificial Systems: An Introductory Analysis With Applications to Biology, Control, and Artificial Intelligence, U Michigan Press, 1975.
[5] 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.
[6] M. Dorigo, V. Dorigo, A. Dorigo, Ant system: optimization by a colony of cooperating agents, IEEE Trans. Syst. Man Cybern. Part B: Cybern. 26 (1) (1996) 29–41.
[7] J. Kennedy, R. Eberhart, Particle swarm optimization, in: Proceedings of IEEE International Conference on Neural Networks, Perth, Australia, 1995, pp. 1942–1948.
[8] R. Eberhart, J. Kennedy, A new optimizer using particle swarm theory, in: Proceedings of the Sixth International Symposium on Micro Machine and Human Science, vol. 1, New York, NY, 1995, pp. 39–43.
[9] D. Karaboga, An Idea Based on Honey Bee Swarm for Numerical Optimization, Tech. Rep., Technical Report-tr06, Erciyes University, Engineering Faculty, Computer Engineering Department, 2005.
[10] Z. Cui, X. Cai, Using social cognitive optimization algorithm to solve nonlinear equations, in: 2010 9th IEEE International Conference on Cognitive Informatics (ICCI), IEEE, 2010, pp. 199–203.
[11] Z. Cui, Z. Shi, J. Zeng, Using social emotional optimization algorithm to direct orbits of chaotic systems, in: Swarm, Evolutionary, and Memetic Computing, Springer, 2010, pp. 389–395.
[12] X. Yang, A new metaheuristic bat-inspired algorithm, in: Nature Inspired Cooperative Strategies for Optimization (NICSO 2010), Springer, 2010, pp. 65–74.
[13] X. Yang, Nature-Inspried Metaheuristic Algorithms, 2nd ed., Luniver Press, Frome, 2010.
[14] Z. Geem, J. Kim, G. Loganathan, A new heuristic optimization algorithm: harmony search, Simulation 76 (2) (2001) 60–68.
[15] D. Simon, Biogeography-based optimization, IEEE Trans. Evol. Comput. 12 (6) (2008) 702–713.
[16] S. He, Q. Wu, J. Saunders, A novel group search optimizer inspired by animal behavioural ecology, in: IEEE Congress on Evolutionary Computation, 2006. CEC 2006, IEEE, 2006, pp. 1272–1278.
[17] W. Gong, Z. Cai, C.X. Ling, De/bbo: a hybrid differential evolution with biogeography-based optimization for global numerical optimization, Soft Comput. 15 (4) (2010) 645–665.
[18] F. Valdez, P. Melin, O. Castillo, An improved evolutionary method with fuzzy logic for combining particle swarm optimization and genetic algorithms, Appl. Soft Comput. 11 (2) (2011) 2625–2632.
[19] X. Yang, S. Deb, Cuckoo search via lévy flights, in: IEEE World Congress on Nature & Biologically Inspired Computing, NaBIC 2009, 2009, pp. 210–214.
[20] X. Yang, S. Deb, Engineering optimisation by cuckoo search, Int. J. Math. Modell. Numer. Optim. 1 (4) (2010) 330–343.
[21] S. Walton, O. Hassan, K. Morgan, M. Brown, Modified cuckoo search: a new gradient free optimisation algorithm, Chaos Solitons Fractals 44 (9) (2011) 710–718.
[22] S. Mishra, Global optimization of some difficult benchmark functions by host-parasite co-evolutionary algorithm, Econ. Bull. 33 (1) (2013) 1–18.
[23] L. Wang, Y. Yin, Y. Zhong, Cuckoo search algorithm with dimension by dimension improvement, Ruan Jian Xue Bao/J. Softw. 24 (11) (2013) 2687–2698.
[24] E. Valian, S. Mohanna, S. Tavakoli, Improved cuckoo search algorithm for global optimization, Int. J. Commun. Inf. Technol. 1 (1) (2011) 31–44.
[25] F. Wang, L. Luo, X. He, Y. Wang, Hybrid optimization algorithm of pso and cuckoo search, in: 2011 2nd International Conference on Artificial

Intelligence, Management Science and Electronic Commerce (AIMSEC), 2011, pp. 1172–1175.
[26] A. Ghodrati, S. Lotfi, A hybrid CS/PSO algorithm for global optimization, in: Intelligent Information and Database Systems, Springer, 2012, pp. 89–98.
[27] X. Li, M. Yin, Parameter estimation for chaotic systems using the cuckoo search algorithm with an orthogonal learning method, Chin. Phys. B 21 (5) (2012) 050507.
[28] X. Li, J. Wang, M. Yin, Enhancing the performance of cuckoo search algorithm using orthogonal learning method, Neural Comput. Appl. (2013) 1–15.
[29] P. Srivastava, R. Khandelwal, S. Khandelwal, S. Kumar, S. Ranganatha, Automated test data generation using cuckoo search and tabu search (CSTS) algorithm, J. Intell. Syst. 21 (2) (2012) 195–224.
[30] G. Wang, L. Guo, H. Duan, L. Liu, H. Wang, B. Wang, A hybrid meta-heuristic DE/CS algorithm for UCAV path planning, J. Inf. Comput. Sci. 5 (2012) (2012) 4811–4818.
[31] A. Layeb, S. Boussalia, A novel quantum inspired cuckoo search algorithm for bin packing problem, Int. J. Inf. Technol. Comput. Sci. 4 (5) (2012) 58–67.
[32] R. Babukartik, P. Dhavachelvan, Hybrid algorithm using the advantage of aco and cuckoo search for job scheduling, Int. J. Inf. Technol. Converg. Serv. 2 (4) (2012) 25–34.
[33] X. Hu, Y. Yin, Cooperative co-evolutionary cuckoo search algorithm for continuous function optimization problems, PR & AI 26 (11) (2013) 1041–1049.
[34] H. Zheng, Y. Zhou, A cooperative coevolutionary cuckoo search algorithm for optimization problem, J. Appl. Math. (2013).
[35] X. Ouyang, Y. Zhou, Q. Luo, H. Chen, A novel discrete cuckoo search algorithm for spherical traveling salesman problem, Appl. Math. Inf. Sci. 7 (2) (2013).
[36] Y. Zhou, H. Zheng, Q. Luo, J. Wu, An improved cuckoo search algorithm for solving planar graph coloring problem, Appl. Math. Inf. Sci. 7 (2) (2013).
[37] M. Marichelvam, An improved hybrid cuckoo search (IHCS) metaheuristics algorithm for permutation flow shop scheduling problems, Int. J. Bio-Inspired Comput. 4 (4) (2012) 200–205.
[38] M. Marichelvam, T. Prabaharan, X. Yang, Improved cuckoo search for hybrid flow shop scheduling problems to minimize makespan, Appl. Soft Comput. 19 (2014) 93–101.
[39] K. Chandrasekaran, S. Simon, Multi-objective scheduling problem: hybrid approach using fuzzy assisted cuckoo search algorithm, Swarm Evol. Comput. 5 (2012) 1–16.
[40] X. Yang, S. Deb, Multiobjective cuckoo search for design optimization, Comput. Oper. Res. 40 (6) (2013) 1616–1624.
[41] D.H. Wolpert, W.G. Macready, No free lunch theorems for optimization, IEEE Trans. Evol. Comput. 1 (1) (1997) 67–82.
[42] W. Long, X. Liang, Y. Huang, Y. Chen, An effective hybrid cuckoo search algorithm for constrained global optimization, Neural Comput. Appl. 25 (3–4) (2014) 911–926.
[43] M. Ballerini, N. Cabibbo, R. Candelier, A. Cavagna, E. Cisbani, I. Giardina, V. Lecomte, A. Orlandi, G. Parisi, A. Procaccini, et al., Interaction ruling animal collective behavior depends on topological rather than metric distance: evidence from a field study, Proc. Natl. Acad. Sci. U. S. A. 105 (4) (2008) 1232–1237.
[44] S. Arya, D.M. Mount, N.S. Netanyahu, R. Silverman, A.Y. Wu, An optimal algorithm for approximate nearest neighbor searching fixed dimensions, J. ACM 45 (6) (1998) 891–923.
[45] Y. Wang, Z. Cai, Q. Zhang, Differential evolution with composite trial vector generation strategies and control parameters, IEEE Trans. Evol. Comput. 15 (1) (2011) 55–66.
[46] M. Reyes-Sierra, C.C. Coello, Multi-objective particle swarm optimizers: a survey of the state-of-the-art, Int. J. Comput. Intell. Res. 2 (3) (2006) 287–308.
[47] S.C. Esquivel, C.A. Coello Coello, On the use of particle swarm optimization with multimodal functions, in: The 2003 Congress on Evolutionary Computation, 2003. CEC'03, vol. 2, IEEE, 2003, pp. 1130–1136.
[48] N. Hansen, A. Ostermeier, Completely derandomized self-adaptation in evolution strategies, Evol. Comput. 9 (2) (2001) 159–195.
[49] P.N. Suganthan, N. Hansen, J.J. Liang, K. Deb, Y.-P. Chen, A. Auger, S. Tiwari, Problem Definitions and Evaluation Criteria for the CEC 2005 Special Session on Real-Parameter Optimization, KanGAL Report 2005005, 2005.
[50] N. Noman, H. Iba, Accelerating differential evolution using an adaptive local search, IEEE Trans. Evol. Comput. 12 (1) (2008) 107–125.
[51] S. García, D. Molina, M. Lozano, F. Herrera, A study on the use of non-parametric tests for analyzing the evolutionary algorithms' behaviour: a case study on the CEC'2005 special session on real parameter optimization, J. Heuristics 15 (6) (2009) 617–644.
[52] W. Gong, Z. Cai, Differential evolution with ranking-based mutation operators, IEEE Trans. Cybern. 43 (6) (2013) 2066–2081.