Applied Mathematical Modelling 63 (2018) 464–490

Contents lists available at ScienceDirect

Applied Mathematical Modelling

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

Queuing search algorithm: A novel metaheuristic algorithm for solving engineering optimization problems

Jinhao Zhang, Mi Xiao, Liang Gao*, Quanke Pan

State Key Laboratory of Digital Manufacturing Equipment and Technology, Huazhong University of Science and Technology, 1037 Luoyu Road, Wuhan, Hubei 430074, China

ARTICLE INFO

Article history:
Received 24 January 2018
Revised 26 May 2018
Accepted 18 June 2018
Available online 11 July 2018

Keywords:
Queuing search algorithm
Metaheuristic algorithms
Engineering optimization

ABSTRACT

This paper presents a novel metaheuristic algorithm called queuing search (QS), which is inspired from human activities in queuing. Some common phenomena are considered in QS: (1) customers actively follow the queue that provides fast service; (2) each customer service is mainly affected by the staff or customer itself; and (3) a customer can be influenced by others during the service when the queue order is not strictly maintained. The performance of QS is tested on 30 bound-constrained benchmark functions scalable with 30 and 100 dimensions from CEC 2014, 5 standard and 4 challenging constrained engineering optimization problems. Meanwhile, comparisons are performed among the results of QS and some state-of-the-art or well-known metaheuristic algorithms.

1. Introduction

Solving optimization problems is an important part of the engineering system design. Up to now, various optimization algorithms have been developed and adopted to search for the optimal solutions of engineering optimization problems. From the point of view of random characteristics of algorithms, these optimization methods can be roughly classified into two categories: conventional deterministic methods and modern metaheuristic algorithms. For deterministic methods, the obtained solution in the kth iteration is determined by that in the (k-1)th iteration. Therefore, the selection of the initial points has a great influence on the final solutions obtained by deterministic methods. Furthermore, deterministic methods are easy to fall into the local optima, and they are not effective in solving high-dimensional, multi-modal and highly constrained engineering optimization problems. As an alternation way, metaheuristic algorithms are preferred by researchers to solve engineering optimization problems. Metaheuristic algorithms differ from deterministic methods in the following three aspects: using a population of individuals in the search domain, using the direct fitness information instead of function derivatives and other related knowledge, and using probabilistic rather than deterministic rules [1]. In metaheuristic algorithms, the realizations of global and local searches are based on two common aspects: exploration and exploitation [2]. The zone containing the best solution is more likely to be achieved by the global exploration ability of metaheuristic algorithms, while the local exploitation ability helps to search for a better solution in this zone. In the beginning of optimization iterations, the exploration ability is utilized to avoid trapping in a local optimum. The exploitation ability has a crucial influence on generating the optimal solutions in the following iterations. To achieve an efficient and accurate search, the tradeoff between exploration and exploitation is crucial.

* Corresponding author.
E-mail address: gaoliang@mail.hust.edu.cn (L. Gao).

https://doi.org/10.1016/j.apm.2018.06.036
0307-904X/

In general, most of metaheuristic algorithms are nature-inspired [2], which can be broadly classified into the following categories in terms of the inspiration source:

(a) Genetic evolution-based algorithms. These algorithms are inspired from biological evolution, such as reproduction, mutation, recombination and selection. Genetic algorithm (GA) [3,4] and differential evolution (DE) [5] are the two most popular algorithms.
(b) Animal-based algorithms. These algorithms mimic the animals' behavior, such as foraging behavior in particle swarm optimization (PSO) [6] and artificial bee colony (ABC) [7], biological migration behavior in biogeography-based optimization (BBO) [8], stealing food behavior in crow search algorithm (CSA) [9].
(c) Plant-based algorithms. These algorithms mimic the behavior of plants. For instance, invasive weed optimization (IWO) algorithm [10] simulates the process of weed invasion. Flower-pollinating algorithm (FPA) [11] imitates the characteristics of flower pollination.
(d) Human activity-based algorithms. These algorithms are inspired from human activities. For example, harmony search (HS) algorithm [12] imitates the musical process of searching for a perfect state of harmony. Teaching-learning based optimization (TLBO) algorithm [13] simulates the process of teacher teaching and student learning. Passing vehicle search (PVS) [14] is developed based on the driving in two-lane highways.
(e) Algorithms inspired from other sources, such as the physics-based algorithm (e.g., water cycle algorithm [15]), and the mathematics-based algorithm (e.g., stochastic fractal search (SFS) algorithm [16]).

Currently, metaheuristic algorithms have gained some applications in real-life problems, such as image processing [17], machine design [18], and scheduling problems [19–22]. However, there is an obstacle to broad applications of these existing metaheuristic algorithms, namely the set of controlling parameters [20]. For example, the most commonly used GA requires adjusting the crossover rate and mutation rate in different application scenarios. Similarly, ABC requires controlling the number of different kinds of bees. PSO requires setting the inertia weight, maximum value of velocity, individual learning factor and social learning factor. In HS, the harmony memory consideration rate, pitch adjusting rate, and number of improvisations need to be predetermined. In CSA, the flight length and awareness probability should be tuned. For these metaheuristic algorithms, a minor change of one parameter value may drastically alter the convergence behavior and the achievable solution quality.

According to the No-Free-Lunch theorem, no single algorithm is suitable for all optimization problems [23]. Therefore, it is valuable to investigate hybrid algorithms, modify existing algorithms and develop new algorithms to achieve better results in real-life applications. In this study, we propose a new metaheuristic algorithm called queuing search (QS), which is inspired from human behaviors in queuing during transacting businesses, shopping, waiting bus, etc. It can be characterized as the aforementioned human activity-based algorithm. In the QS algorithm, except for the population size and the stopping criterion, which are two common parameters in all metaheuristic algorithms, no other parameters need to be set in its applications. To check the performance of the QS algorithm, 30 bound-constrained benchmark functions scalable with 30 and 100 dimensions from CEC 2014 [24] are tested. Then, the proposed QS algorithm is further tested on five standard and four challenging constrained engineering optimization problems. Through the comparison of QS and some state-of-the-art or well-known metaheuristic algorithms, it is demonstrated that QS is a highly competitive metaheuristic algorithm and has the great superiority and competitiveness on jumping out of a local optimal solution and searching the global optimum.

2. Queuing search algorithm

2.1. Background

Queuing behavior is a common situation in our daily life, such as queuing to transact businesses, check out after shopping and take bus. Customers can be served by staffs in different queues, but they often choose the queue where they can get service in the shortest time. In addition, each staff often serves the different number of customers within the same amount of time due to their different personal business capabilities. Generally, the staffs with stronger business capabilities will serve more customers. Each customer service is mainly affected by the staffs or the customer itself. Also, other customers may have an influence on this customer service when the queue order is not strictly maintained.

2.2. Principles in QS

Inspired from the aforementioned queuing process, a novel metaheuristic algorithm called as queuing search (QS) is proposed in this paper. Five principles in the QS algorithm are listed as follows:

Principle 1: The number of customers is limited and fixed in the queuing. This principle is interpreted that the number of individuals used in searching for the optimal solution is preset and fixed in QS.

Principle 2: Some businesses must be handled while the others are optional. For example, in physical examination, the basic examinations are necessary but some special examinations are optional. This principle means that some update procedures for individuals are necessary while the others are selectable in QS.

Principle 3: The staffs with stronger business capabilities will serve more customers within the same amount of time. After each iteration in QS, the first few best individuals (FFBIs) are judged according to the fitness of each individual. Each

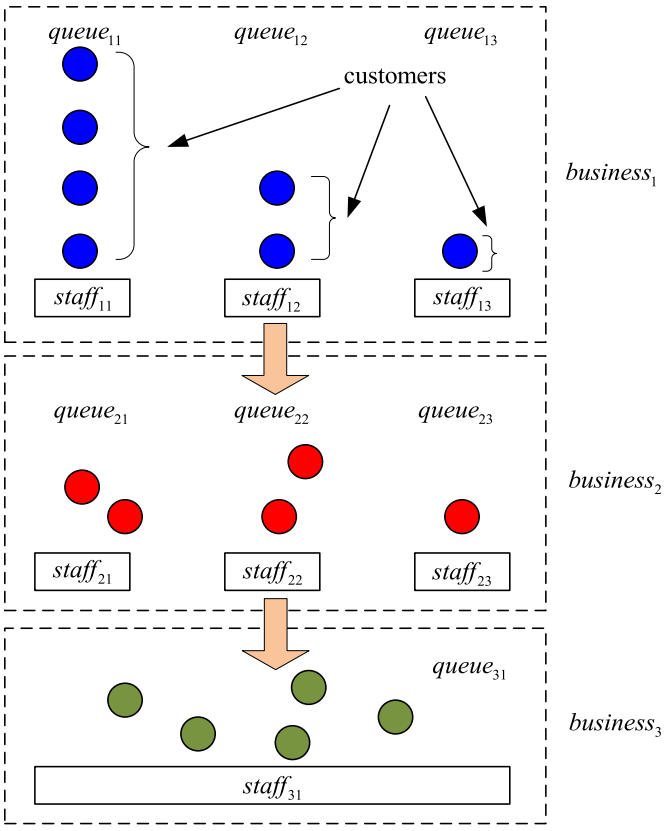

Fig. 1. The multi-staff queuing system.

of FFBIs will determine a local search region. This principle means that the local search region determined by one of FFBIs with better fitness will be distributed more individuals in QS.

Principle 4: Each customer service is mainly dominated by the customer itself or the staffs. This principle means that the update of an individual can be realized based on itself or the FFBIs in QS.

Principle 5: A fluctuation of each customer service can be caused by the customer itself, the staffs and other customers. For example, failure to strictly maintain the queue order will cause a certain degree of disorder. As the disorder deteriorates, the influence of other customers on one customer service will be increased. Ultimately, this customer service will be mainly affected by other customers. This principle means that the fluctuation information about the update direction and distance of each individual in QS can be generated from the individual itself, the FFBIs and other individuals. Under some special circumstances, other individuals can have a crucial impact on the update of one individual.

2.3. A queuing system in QS

In this research, a multi-staff queuing system with three types of businesses is established and shown in Fig. 1. The QS algorithm is realized by simulating this queuing system based on the aforementioned principles. The components of this queuing system are fixed and not required to be adjusted in QS except the number of customers. The detailed introduction about this queuing system is given in the following.

To embody the *Principle 5*, three cases with different degrees of disorder are considered. Firstly, the order of each queue is strictly maintained, which is used in $business_1$. Secondly, the order of each queue is not strictly maintained, which is applied in $business_2$. Finally, the queue is completely disordered, which is employed in $business_3$. Thus, three businesses are defined in the simplified queuing system. Because it is demonstrated that customers tend to follow quicker queues in *Principle 3*, multiple staffs are required in the simplified queuing system, and three staffs is enough to embody this principle. Actually, more staffs can be also chosen, but it is not necessary and will increase the difficulty of realizing the algorithm. Therefore, three staffs are defined in $business_1$ and $business_2$. One staff is set in $business_3$ due to only one completely disordered queue defined in this business.

Based on *Principle 1*, it is assumed that the total number of customers is N. The ith customer is denoted by $customer_i$. The $customer_i$'s state is denoted by a vector $\boldsymbol{X}_i$ where $\boldsymbol{X}_i = [X_{i,1}, X_{i,1}, \ldots, X_{i,D}]$, and f_i represents the complexity of $customer_i$'s state. Let the vector $\boldsymbol{A}_{mn}$ with D dimensions denotes $staff_{mn}$'s ability to handle the customers' businesses, where m represents the

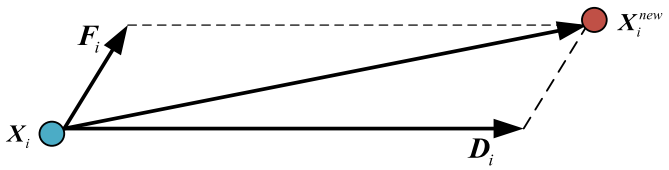

Fig. 2. Update process of customer's state.

type of business and n is used to distinguish different staffs in the same business. $queue_{mn}$ represents the queue in front of $staff_{mn}$. T_{mn} denotes the potential service time of $staff_{mn}$ for each customer to deal with the $business_m$. The less the T_{mn} is, the more customers the $staff_{mn}$ may serve. It is assumed that the T_{mn} is required to satisfy $0 \le T_{11} \le T_{12} \le T_{13}$ and $0 \le T_{21} \le T_{22} \le T_{23}$. According to *Principle 4*, two different dominators are considered during the process of customer service in this system: customer-dominator and staff-dominator. Based on *Principle 5*, fluctuation is considered in the service process. In this queuing system, the service process is represented by the update of customers' state as follows:

$$\mathbf{X}_i^{new}=\mathbf{D}_i+\mathbf{F}_i \tag{1}$$

where $\mathbf{X}_i^{new}$represents the updated state of $customer_i$, $\boldsymbol{D}_i$ denotes the dominator in the service process of $customer_i$. And $\boldsymbol{F}_i$ represents the fluctuation of service process of $customer_i$. The update process is presented in Fig. 2.

Erlang distribution is popularly applied in queuing theory. In this study, the random number for fluctuation in *Principle 5* is produced based on the Erlang distribution. The probability density function (PDF) of a k-order Erlang distribution is formulated in the following equation:

$$f(t)=\begin{cases}\frac{k\mu(k\mu t)^{k-1}}{(k-1)!}\mathrm{e}^{-k\mu t} & ,t>0\\ 0 & ,t\le 0\end{cases} \tag{2}$$

where k is the order of the Erlang distribution and μ is the frequency of an event. Because the random number is independently generated in one business, k is set to 1. The mean value of the Erlang distribution is $k\mu$. Then μ is set to 0.5 in order to obtain the Erlang distribution with the mean value 0.5. Under these settings of k and μ in the Erlang distribution, the probability of simulating a random number in the interval [0, 1] will reach 86.5%.

2.3.1. Queuing process in business_1

In $business_1$, there are three queues denoted by $queue_{1n}$ (n = 1, 2, 3). For *Principle 2*, all the customers are required to handle this business. Based on *Principle 3*, the number q_{1n} of customers in $queue_{1n}$ is inversely proportional to the potential service time T_{1n} of $staff_{1n}$, which can be calculated as follows:

$$q_{1n}=N\frac{\frac{1}{T_{1n}}}{\frac{1}{T_{11}}+\frac{1}{T_{12}}+\frac{1}{T_{13}}},\ n=1,2,3. \tag{3}$$

For *Principle 5*, it is assumed that the order of each queue in $business_1$ is strictly maintained, which means that customers have no mutual influence on each other in this business. Then two state update patterns of $customer_i$ are defined and represented as follows:

$$\mathbf{X}_i^{new}=\mathbf{A}+\mathbf{F}_{11},\ i=1,2,...,N \tag{4}$$

$$\mathbf{X}_i^{new}=\mathbf{X}_i+\mathbf{F}_{12},\ i=1,2,...,N \tag{5}$$

where $\boldsymbol{A}$ is equal to $\boldsymbol{A}_{1n}$ when $customer_i$ is in $queue_{1n}$ (n = 1, 2, 3). For *Principle 5*, it is assumed that the order of each queue in $business_1$ is strictly maintained, which means that customers have no mutual influence on each other in this business. The update fluctuation can be randomly caused by the customer itself and the corresponding staff. Considering the matching difference between the customer's state and the business ability of the staff, the fluctuation can be also generated from the difference between ability $\boldsymbol{A}$ and state $\boldsymbol{X}_i$, namely $\boldsymbol{A}$–$\boldsymbol{X}_i$. Fluctuation $\boldsymbol{F}_{11}$ and $\boldsymbol{F}_{12}$ are defined and represented as

$$\mathbf{F}_{11}=\beta\times\alpha\times(\mathbf{E}\circ|\mathbf{A}-\mathbf{X}_i|)+(e\times\mathbf{A}-e\times\mathbf{X}_i) \tag{6}$$

$$\mathbf{F}_{12}=\beta\times\alpha\times(\mathbf{E}\circ|\mathbf{A}-\mathbf{X}_i|) \tag{7}$$

where α is a random number and generated from [−1,1] with equal probability, which is used to control the direction of fluctuation. $\boldsymbol{E}$ is a D dimensional vector and consists of D random numbers, which are generated based on the Erlang distribution. e is also a random number generated based on the Erlang distribution. The symbol $\circ$ represents the dot product, and | | represents the absolute value of a number. β is utilized to adjust the range of fluctuation. In QS, β can be defined

based on the stopping criterion. If the maximum number of generations ($iter_{max}$) is chosen as the stopping criterion, β can be calculated as follows:

$$\beta = e^{(\ln\frac{1}{g})\times(\frac{g}{iter_{max}})^{0.5}} \tag{8}$$

where g is the current iteration number. It can be found that β will reduce as the increase of g. If the maximum number of function evaluations (MFES) is chosen as the stopping criterion, β can be calculated as follows:

$$\beta = e^{(\ln\frac{1}{g})\times(\frac{FES}{MFES})^{0.5}} \tag{9}$$

where *FES* is the current number of function evaluations. Because *FES*/MFES is close to $g/iter_{max}$, the update process of β represented by Eq. (9) is similar to that by Eq. (8).

In Eq. (4), $\boldsymbol{A} = \boldsymbol{A}_{1n}$ represents that $staff_{1n}$ dominates the service process of $customer_i$. In QS, the use of $\boldsymbol{A}_{1n}$ means that the update of an individual is based on one of the FFBIs. In Eq. (5), $\boldsymbol{X}_i$ denotes that the service process of $customer_i$ is dominated by itself, which means that the update of an individual depends on itself in QS. Because the order of each queue in $business_1$ is strictly maintained, the fluctuation of the service process of $customer_i$ is introduced by $staff_{1n}$ and $customer_i$ itself. As shown on the right side of Eqs. (4) and (5), fluctuation items are added into the two update expressions of $customer_i$' state, respectively. In QS, the fluctuation items represent the update direction and distance of each individual.

In $business_1$, each customer can only choose either of the two state update patterns. The state of the first customer in a queue is updated by Eq. (4). The state update pattern of $customer_i$ is determined by that of $customer_{i-1}$. Specially, if the updated complexity of $customer_{i-1}$'s state is reduced, the same update pattern will be used for $customer_i$. Otherwise, the other update pattern will be chosen by $customer_i$. After handling $business_1$, all the customers will queue for the following $business_2$.

2.3.2. Queuing process in $business_2$

In $business_2$, there are three queues denoted by $queue_{2n}$ (n = 1, 2, 3). For *Principle 2*, a part of customers need to handle this business. In this business, $customer_i$ obtains a probability Pr_i by the following equation:

$$Pr_i = \frac{rank(f_i)}{N}, i = 1, 2, ..., N \tag{10}$$

where $rank(f_i)$ denotes the ranking of the f_i of $customer_i$ when the state complexity of all the customers is ranked in descending order. To determine whether $customer_i$ handles the $business_2$, a random number is generated from [0, 1] based on uniform distribution. If the random number is less than Pr_i, $customer_i$ will handle the $business_2$. This operation is used to increase the chance of accepting $business_2$ for $customer_i$ with larger Pr_i. According to *Principle 3*, the number q_{2n} of customers in $queue_{2n}$ can be calculated in the same way as in $business_1$:

$$q_{2n} = N\frac{\frac{1}{T_{2n}}}{\frac{1}{T_{21}}+\frac{1}{T_{22}}+\frac{1}{T_{23}}}, \quad n = 1, 2, 3. \tag{11}$$

For *Principle 4*, only customer-dominance is considered in this business. Then two state update patterns of $customer_i$ in $business_2$ are defined and represented as follows:

$$\mathbf{X}_i^{new} = \mathbf{X}_i + \mathbf{F}_{21}, i = 1, 2, ..., N_2 \tag{12}$$

$$\mathbf{X}_i^{new} = \mathbf{X}_i + \mathbf{F}_{22}, i = 1, 2, ..., N_2 \tag{13}$$

where N_2 denotes the number of customers which need to handle $business_2$.

For *Principle 5*, it is assumed that the order of each queue in $business_2$ is not strictly maintained. Other customers can cause fluctuation in the update process of customer's state. To simplify the fluctuation, two random selected customer are consider to cause the fluctuation, namely $\mathbf{X}_{r1} - \mathbf{X}_{r2}$. Sometimes, other customers and the staff can generate the hybrid fluctuation, namely $\mathbf{A} - \mathbf{X}_{r1}$. Then $\boldsymbol{F}_{21}$ and $\boldsymbol{F}_{22}$ are defined and represented as

$$\mathbf{F}_{21} = e \times (\mathbf{X}_{r1} - \mathbf{X}_{r2}) \tag{14}$$

$$\mathbf{F}_{22} = e \times (\mathbf{A} - \mathbf{X}_{r1}) \tag{15}$$

where $\boldsymbol{A}$ is equal to $\boldsymbol{A}_{2n}$ when $customer_i$ is in $queue_{2n}$ (n = 1, 2, 3). $\boldsymbol{X}_{r1}$ and $\boldsymbol{X}_{r2}$ are the states of two randomly selected customers. The choice of these two state update patterns depends on the value of a confusion degree c_v, which is defined as follows:

$$c_v = \frac{T_{21}}{T_{22} + T_{23}}. \tag{16}$$

From Eq. (16), it can be seen that $c_v \in [0, 0.5]$. For $customer_i$, a random number is generated from [0, 1] based on uniform distribution. If this random number is less than c_v, $customer_i$ will employ the state update pattern in Eq. (12). Otherwise, Eq. (13) will be used. As the increase of c_v, the probability of utilizing Eq. (12) will be raised.

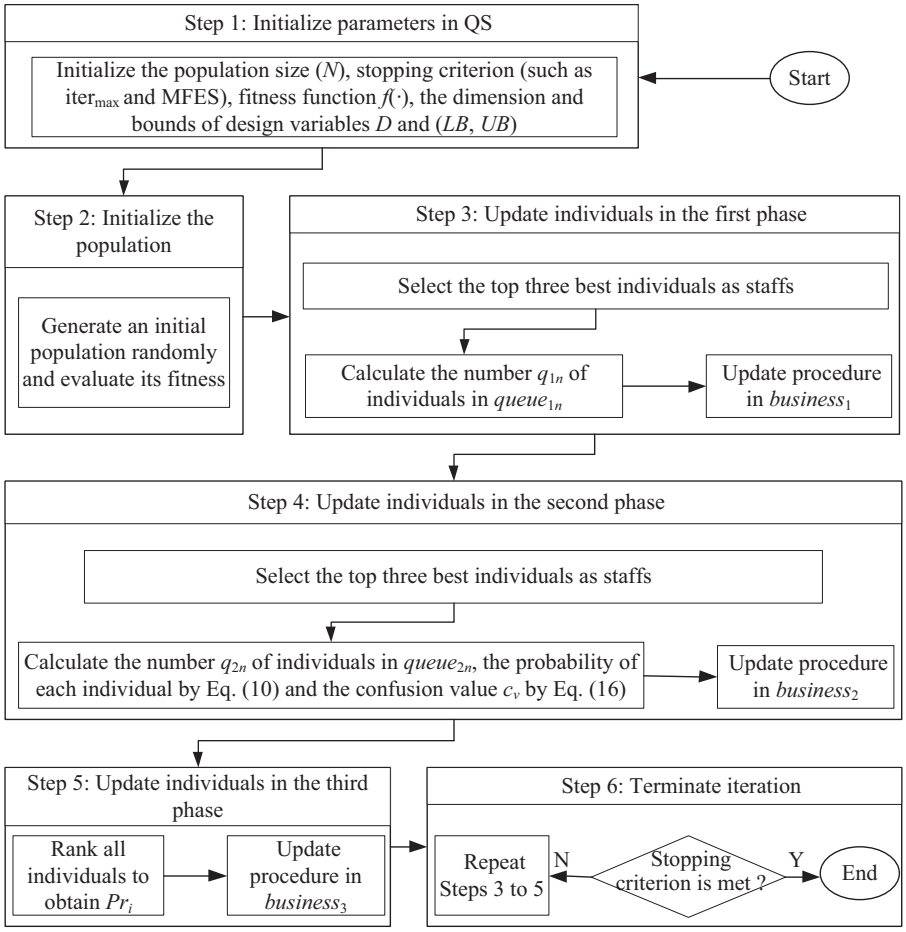

Fig. 3. Flowchart of the QS algorithm.

2.3.3. Queuing process in business$_3$

In *business*$_3$, a part of customers are required to handle this business for *Principle 2*. The state of a customer is updated in dimension in this business. For the *d*th dimension of *customer*$_i$'s state, a random number is generated from [0, 1] based on uniform distribution. If this random number is larger than Pr_i evaluated by Eq. (10), the $\underline{d}$th dimension of *customer*$_i$'s state will be updated in *business*$_3$. It can be seen that if *customer*$_i$ has a small Pr_i, each dimension of *customer*$_i$'s state will have more chances to handle *business*$_3$. For *Principle 5*, it is assumed that the queue in *business*$_3$ is completely disordered, which means that the mutual influence of customers is so serious in this business that the service for a customer is mainly affected by other customers. Therefore, the update process is determined by other customers, and fluctuation is caused by customer itself and other customers. Then the state update pattern of *customer*$_i$ in *business*$_3$ is defined and described as

$$\mathbf{X}_i^{new}=\mathbf{X}_{r1,d}+\left(e\times\mathbf{X}_{r2,d}-e\times\mathbf{X}_{i,d}\right),\ i=1,2,\ldots,N_3 \tag{17}$$

where N_3 denotes the number of customers which need to handle *business*$_3$. $\boldsymbol{X}_{r1}$ and $\boldsymbol{X}_{r2}$ are the states of two randomly selected customers. From Eq. (17), it can be observed that other customers have a great influence on the service process of *customer*$_i$.

3. Implementation of QS for optimization

To help understand how the QS algorithm simulates the queuing system, the correspondences between them are elaborated. Customers correspond to individuals. The state vector $\boldsymbol{X}_i$ of *customer*$_i$ denotes the solution of individual *i*. The complexity of *customer*$_i$'s state f_i is considered as the fitness of individual *i*. The three staffs in *business*$_1$ *and business*$_2$ correspond to FFBIs in the current iteration. The different businesses represent different update phases, and each state update pattern denotes a special individual update strategy. In the following, a step-wise procedure for implementation of QS is given and QS is explained with the aid of the flowchart in Fig. 3.

Step 1: Initialize the algorithm parameters and define the optimization problem.

Algorithm 1
Pseudo code of update procedure in $business_1$.

```
for i = 1:N
  if i ≤ q11
    if i = 1; case = 1; end if
    A = A11
  else if i > q11 and i≤ q11 + q12
    if i = q11 + 1; case = 1; end if
    A = A12
  else
    if i = q11 + q12 + 1; case = 1; end if
    A = A13
  end if
  Update β
  if case = 1
    Obtain X_i^new based on Eq. (4)
    if X_i^new obtains a better fitness function values
      Replace X_i with X_i^new
      case = 1;
    else
      case = 2;
    end if
  else
    Obtain X_i^new based on Eq. (5)
    if state_i^new obtains a better fitness function values
      Replace X_i with X_i^new;
      case = 2;
    else
      case = 1;
    end if
  end if
end for
```

QS is initialized with the population size N, stopping criterion (such as $\text{iter}_{\max}$ and MFES), fitness function $f(\cdot)$, the dimension of design variables (D), and the upper and lower bounds of design variables (LB, UB).

Step 2: Initialize the population.

A random population is generated on the basis of N, D, LB and UB. An initial solutions in the population can be described by

$$\mathbf{X}_{1\ldots N,\,1\ldots D} = \begin{bmatrix} X_{1,1} & X_{1,2} & \ldots & X_{1,D} \\ X_{2,1} & X_{2,2} & \ldots & X_{2,D} \\ \vdots & \vdots & & \vdots \\ X_{N,1} & X_{N,2} & \ldots & X_{N,D} \end{bmatrix} \tag{18}$$

Subsequently, the fitness of individual i is calculated by $f_i = f(\boldsymbol{X}_i)$.

Step 3: Update solutions in the first phase.

Based on the fitness, all individuals are ranked in descending order, and the top three best individuals are selected as the three staffs. Then, the number q_{1n} of individuals in $queue_{1n}$ is calculated by Eq. (3). Let $case = 1$ and $case = 2$ represent the use of the update patterns by Eqs. (4) and (5), respectively. Subsequently, the update procedure in the first phase $business_1$ is shown in Algorithm 1.

Step 4: Update solutions in the second phase.

All individuals are sorted in descending order based on their fitness. Similar to the first update phase, the top three best individuals are selected as the three staffs. Each individual is assigned a probability Pr_i by Eq. (10). The number q_{2n} of individuals in $queue_{2n}$ is gained by Eq. (11). The confusion degree c_v is calculated by Eq. (16). The update procedure in the second phase $business_2$ is given in Algorithm 2.

Step 5: Update solutions in the third phase.

All individuals are sorted in descending order based on their fitness. Each individual is assigned a probability Pr_i by Eq. (10). The update procedure in the third phase $business_3$ is presented in Algorithm 3.

Step 6: Terminate iteration.

Steps 3–5 are repeated until the termination criterion is satisfied.

4. Analysis of QS

Exploration and exploitation are two significant features in metaheuristic algorithms. In QS, exploration and exploitation are realized by some principles inspired from the queuing process:

Algorithm 2
Pseudo code of update procedure in $business_2$.

```
for i = 1:N
  if i ≤ q21
    A = A21
  else if i > q21 and i≤ q21 + q22
    A = A22
  else
    A = A23
  end if
  if rand < Pri
    Randomly select two individuals Xr1 and Xr2
    if rand < cv
      Obtain Xi^new based on Eq. (12)
    else
      Obtain Xi^new based on Eq. (13)
    end if
    Accept Xi^new if it obtains a better fitness function value
  end if
end for
```

Algorithm 3
Pseudo code of update procedure in $business_3$.

```
for i = 1:N
  for d = 1:D
    if rand > Pri
      Randomly select two individuals Xr1 and Xr2
      Obtain Xi^new based on Eq. (17)
    end if
  end for
  Accept Xi^new if it obtains a better fitness function value
end for
```

In *Principle 3*, different numbers of individuals are assigned to the first three optimal solutions and follow them, respectively. In the initial iterations, the first three optimums are likely to be scattered in different local regions to achieve the global exploration. However, it is likely that they are located in the same local region with the potential global optimum in the last stage of iteration, and the local exploitation is performed to search for the optimal solution. So the exploration and exploitation can be achieved based on the distribution characteristics of the first three optimums in their current iteration.

In *Principle 4*, the update strategy dominated by FFBIs enhances the local exploitation in QS, while the update strategy dominated by individuals provides the chance of searching in the global space. Furthermore, the adaptive change of β in Eqs. (6) and (7) adjusts exploration and exploitation by controlling the range of fluctuation. As the iteration proceeds, β will reduce. The range of search will become narrow and the local exploitation will be enhanced.

In addition, in order to avoid premature convergence, the adaptive factor c_v in Eq. (16) is utilized to control the update of individuals in the second phase $business_2$. When the first few optimal solutions fall into one local region, T_{21} will be approximately equal to T_2 and T_{23} and c_v will tend to maximum value 0.5, which increases the chance of utilizing the update pattern based on Eq. (12). Because Eq. (12) does not use the information of the current optimal solution, a larger c_v will increase the ability of QS to jump out of the local optima. In $business_3$, for the dth dimension of an individual, it is updated with the information of randomly selected two individuals. The information of the current optimal solution is not used. This update pattern of individuals can also increase the global search ability of QS.

Like many other metaheuristic algorithms, such as GA, PSO, ABC, TLBO, PVS and CSA, QS is a population-based algorithm and adopts a group of individuals to search the optimal solution. The parameters in each algorithm have a great influence on the performance of the algorithm, and it is time-consuming to adjust these parameters for different applications of each algorithm. However, except the initial population size and the stop criteria, QS does not need other parameters to be preset in its applications.

Similar to DE and TLBO, QS utilizes the information of the current best solution to update some solutions. Moreover, QS also employ the information of other individuals to perform the update of an individual. Like ABC with employed and onlooker bee phases and TLBO with teaching and learning phases, three different business phases are used to update solutions in QS.

5. Formulations of constrained optimization problems

In this study, bound-constrained and common constrained optimization problems are selected to test the performance of the proposed QS algorithm. For the bound-constrained problems, generally, each design variable only needs to satisfy a

Table 1
Descriptions of 30 benchmark functions.

Groups	No.	Benchmark functions	Search range	Global optimum
Unimodal functions	f_1	Rotated high conditioned elliptic function	[−100, 100]	100
	f_2	Rotated bent cigar function	[−100, 100]	200
	f_3	Rotated discus function	[−100, 100]	300
Multimodal functions	f_4	Shifted and rotated Rosenbrock's function	[−100, 100]	400
	f_5	Shifted and rotated Ackley's function	[−100, 100]	500
	f_6	Shifted and rotated Weierstrass function	[−100, 100]	600
	f_7	Shifted and rotated Griewank's function	[−100, 100]	700
	f_8	Shifted Rastrigin's function	[−100, 100]	800
	f_9	Shifted and rotated Rastrigin's function	[−100, 100]	900
	f_{10}	Shifted Schwefel's function	[−100, 100]	1000
	f_{11}	Shifted and rotated Schwefel's function	[−100, 100]	1100
	f_{12}	Shifted and rotated Katsuura function	[−100, 100]	1200
	f_{13}	Shifted and rotated HappyCat function	[−100, 100]	1300
	f_{14}	Shifted and rotated HGBat function	[−100, 100]	1400
	f_{15}	Shifted and rotated expanded Griewank's plus Rosenbrock's function	[−100, 100]	1500
	f_{16}	Shifted and rotated expanded Scaffer's F6 function	[−100, 100]	1600
Hybrid functions	f_{17}	Hybrid function 1 ($N = 3$)	[−100, 100]	1700
	f_{18}	Hybrid function 2 ($N = 3$)	[−100, 100]	1800
	f_{19}	Hybrid function 3 ($N = 4$)	[−100, 100]	1900
	f_{20}	Hybrid function 4 ($N = 4$)	[−100, 100]	2000
	f_{21}	Hybrid function 5 ($N = 5$)	[−100, 100]	2100
	f_{22}	Hybrid function 6 ($N = 5$)	[−100, 100]	2200
Composition functions	f_{23}	Composition function 1 ($N = 5$)	[−100, 100]	2300
	f_{24}	Composition function 2 ($N = 3$)	[−100, 100]	2400
	f_{25}	Composition function 3 ($N = 3$)	[−100, 100]	2500
	f_{26}	Composition function 4 ($N = 5$)	[−100, 100]	2600
	f_{27}	Composition function 5 ($N = 5$)	[−100, 100]	2700
	f_{28}	Composition function 6 ($N = 5$)	[−100, 100]	2800
	f_{29}	Composition function 7 ($N = 3$)	[−100, 100]	2900
	f_{30}	Composition function 8 ($N = 3$)	[−100, 100]	3000

boundary constraint:

$$LB_i \le x_i \le UB_i, i = 1, 2, ..., l \tag{19}$$

where LB_i and UB_i are the lower and upper bounds of the design variable x_i, and l is the number of design variables. On the other hand, a common constrained optimization problem can be typically formulated as

$$\begin{aligned} &\min \quad f(\boldsymbol{X}) \\ &\quad \boldsymbol{X} = \{x_1,\ x_2,\ \ldots,\ x_l\} \\ &\text{s.t.} g_j(\boldsymbol{X}) \le 0, \quad j = 1, 2, \ \ldots, m \\ &\quad h_k(\boldsymbol{X}) = 0, \quad k = 1, 2, \ \ldots, n \\ &\quad LB_i \le x_i \le UB_i, i = 1, 2, ..., l \end{aligned} \tag{20}$$

where m is the number of inequality constraints and n is the number of equality constraints.

In the performance test of the QS algorithm in this work, all the constrained problems in Eq. (20) are converted into bound-constrained problems by using the static penalty approach [25]. For each infeasible solution, a penalty value will be added into the objective function. Due to its simplicity and convenience in applications, the static penalty approach is popular. It only requires an auxiliary penalty and is suitable for all different problems [14]. Using this approach, the constrained problem in Eq. (20) can be rewritten as

$$f(\boldsymbol{X}) = f(\boldsymbol{X}) + \sum_{j=1}^{m} Pe_j \max\{g_j(\boldsymbol{X}), 0\} + \sum_{k=1}^{n} Pe_k \max\{|h_k(\boldsymbol{X})| - \varepsilon, 0\} \tag{21}$$

where Pe_j and Pe_k are penalty factors and usually assigned a large value. ε is the error of equality constraints, which is set to 1e−6 in this study.

6. Test of benchmark functions

To check the performance of the QS algorithm, 30 bound-constrained benchmark functions from CEC 2014 are chosen. These functions span various characteristics, such as ruggedness, noise in fitness, multimodality, ill-conditioning, interdependence and non-separability. To summarize simply, functions f_1–f_3 are unimodal, functions f_4–f_{16} are multimodal, functions f_{17}–f_{22} are hybrid, and functions f_{23}–f_{30} are composition functions. Table 1 shows all these functions. More details on these functions can be referred in [24]. The test scheme is given in the following sections.

Table 2
Parameter settings for all the algorithms in comparison.

Algorithms	Parameter settings
PSO	Inertia weight $w = 1$, acceleration coefficients $c_1 = c_2 = 2$, and population size = 100 [6].
GA	No elitism is used. Mutation probability = 0.05, crossover probability = 0.95, and population size = 100 [3].
ABC	Limit = 50D, and population size = 100 [7].
SFS	Maximum diffusion number = 1, Gaussian walk = 0, and population size = 100 [16].
BBO	Habitat modification probability = 1, immigration probability bounds per gene = [0, 1], step size for numerical integration of probabilities = 1, and population size = 100 [8].
APSO	The inertia w is initialized to 0.9, and then is adaptively adjusted. Accelerations coefficients c_1 and c_2 are both initialized to 2.0, and then is adaptively controlled. The number of particles is set to 20 [26].
OLPSO	The inertia w is initialized to 0.9, and then is linearly decreased to 0.4. Acceleration coefficient $c = 2$, reconstruction gap $G = 5$, and the number of particles is set to 40 [27].
DVBA	The number of wave vectors $w = 6$, the number of search positions $m = 5$, increment rate divisor $\beta = 100$, and population size = 100 [28].
EPSDE	Population size = 50 [29].
JADE	Population size = 100 [30].
SaDE	Population size = 30 [31].
FWA-DM	Parameter settings of these algorithms are the same as those in the original papers [32,33].
NRGA	
LSHADE44	Parameter settings of the algorithm are the same as those in its original paper, and the population is linearly decreased to 4.0 starting from $18 \times D$ [37].
b3e3pbest	Parameter settings of these algorithms are the same as those in the original papers [34–36].
OPTBees	
b636rl	
QS	Population size = 100.

6.1. Comparison algorithms and parameter settings

For comparison, 17 metaheuristic algorithms are chosen and classified into three groups. In the first group, to make a preliminary comparison, some standard algorithms are compared with the QS algorithm on 30-D test functions. These algorithms include the state-of-the-art SFS algorithm and some well-known algorithms: PSO, GA, ABC and BBO. To make a further comparison, in the second group, the QS algorithm is compared with some state-of-the-art modified algorithms on 30-D test functions. The modified algorithms include APSO [26], OLPSO [27], DVBA [28], EPSDE [29], JADE [30] and SaDE [31]. In the third group, to check the performance of the QS algorithm in solving high dimensional optimization problems, comparisons are made on 100-D test functions and some state-of-the-art algorithms are selected: FWA-DM [32], NRGA [33], b3e3pbest [34], OPTBees [35] and b636rl [36] from the special session at CEC 2014, and LSHADE44 [37] from the special session at CEC 2016. The parameter settings for all the algorithms in comparison are shown in Table 2.

6.2. Performance measures

In the comparison, the mean error (MeanErr) and standard deviation (StdDev) are employed to evaluate the performance of each algorithm, which can be calculated by

$$\text{MeanErr} = \frac{1}{N}\sum_{i=1}^{N}[f(\mathbf{X}_i) - f(\mathbf{X}^*)] \tag{22}$$

$$\text{StdDev} = \sqrt{\frac{1}{N}\sum_{i=1}^{N}[f(\mathbf{X}_i) - f(\mathbf{X}^*)]^2} \tag{23}$$

where N is the number of independent runs, $\boldsymbol{X}_i$ is the best searched value in the ith run and $\boldsymbol{X}^*$ is the global optimum for the test function $f(\boldsymbol{X})$. $f(\boldsymbol{X}_i)$–$f(\boldsymbol{X}^*)$ represents the optimization error of an algorithm in an independent run. The smaller mean error is, the stronger the search capability of an algorithm is. The smaller standard deviation is, the more robust an algorithm is. The smallest mean error values will be highlighted in boldface in the following sections.

Furthermore, two-tailed t-tests [38] are used in this study to compare the mean errors obtained by QS and the other algorithms at the 0.05 level of significance. The statistical significance level of the aggregate results is given in the last three rows of the Tables 3–6. "–" signifies the case where a compared algorithm shows a poor performance than QS. "+" means that a compared algorithm presents a better performance than QS. "≈" indicates that a compared algorithm and QS are not significantly different.

6.3. Comparison 1: QS and five standard algorithms

In this section, 30 bound-constrained benchmark functions with 30 dimensions from CEC 2014 shown in Table 1 are employed for the comparison of QS and five standard algorithms: PSO, GA, ABC, SFS and BBO. For each test function, the mean

Table 3
Comparisons of PSO, GA, ABC, SFS, BBO and QS over 30 test functions with 30 dimensions.

Func.	PSO			GA			ABC			SFS			BBO			QS	
	MeanErr	StdDev	*t*-test	MeanErr	StdDev	*t*-test	MeanErr	StdDev	*t*-test	MeanErr	StdDev	*t*-test	MeanErr	StdDev	*t*-test	MeanErr	StdDev
*f*1	5.41E+07	2.24E+07	–	1.09E+06	5.87E+05	–	6.11E+08	1.26E+08	–	8.98E+04	7.38E+04	–	1.95E+06	8.26E+05	–	**3.72E+03**	7.17E+03
*f*2	2.42E+08	4.23E+08	–	7.57E+06	2.51E+04	–	8.57E+07	8.25E+07	–	1.37E+02	5.12E+01	–	5.69E+04	2.07E+04	–	**9.76E–04**	7.27E–04
*f*3	5.97E+04	1.07E+04	–	2.14E+04	8.27E+03	–	2.73E+05	6.67E+04	–	1.91E+00	9.58E–01	–	1.35E+03	1.66E+03	–	**7.90E–05**	3.62E–05
*f*4	1.66E+02	8.29E+01	–	3.62E+00	9.54E–01	–	2.77E+01	2.37E–01	–	2.78E+01	3.22E+01	–	7.60E+01	4.64E+01	–	**2.15E–01**	4.34E–01
*f*5	2.08E+01	9.01E+01	–	2.09E+01	6.84E–02	–	2.10E+01	4.46E–02	–	2.06E+01	5.83E–02	–	**2.00E+01**	1.58E–02	+	2.04E+01	5.23E–02
*f*6	2.58E+01	6.40E+01	≈	2.15E+01	1.06E+00	–	3.93E+01	8.87E–01	–	1.42E+01	2.83E+00	–	1.27E+01	4.29E+00	–	**4.88E+00**	3.07E+00
*f*7	9.28E+01	1.23E+01	–	1.28E+00	1.24E–02	–	2.65E–01	6.47E–02	–	**6.03E–05**	2.52E–04	≈	2.04E–01	7.46E–02	–	4.11E–04	2.25E–03
*f*8	**5.09E–02**	1.11E+00	≈	1.01E+00	9.40E+00	–	2.01E+02	1.08E+01	–	6.17E+00	1.51E+00	–	2.59E+01	7.02E+00	–	3.65E–01	6.65E–01
*f*9	3.18E+01	1.28E+02	≈	**2.83E+01**	6.37E+00	+	2.21E+02	1.16E+01	–	6.04E+01	7.48E+00	–	4.95E+01	1.43E+01	≈	5.28E+01	1.26E+01
*f*10	7.61E+02	7.61E+02	–	8.85E+02	4.27E+02	–	7.15E+03	2.79E+02	–	9.30E+01	2.55E+01	–	1.01E+03	4.22E+02	–	**2.46E+00**	1.90E+00
*f*11	6.83E+03	1.93E+03	–	7.53E+03	4.25E+02	–	7.70E+03	2.36E+02	–	2.65E+03	2.44E+02	–	3.01E+03	6.11E+02	–	**1.79E+03**	4.78E+02
*f*12	2.86E+00	2.77E+01	–	5.86E–01	6.81E–02	–	2.48E+00	3.39E–01	–	7.70E–01	1.15E–01	–	**1.48E–01**	6.39E–02	+	3.93E–01	6.83E–02
*f*13	6.40E–01	4.44E–01	–	2.79E–01	6.24E–03	≈	4.97E–01	7.15E–02	–	2.85E–01	5.14E–02	≈	**2.15E–01**	5.52E–02	+	2.65E–01	5.00E–02
*f*14	4.21E–01	1.57E–01	–	2.54E–01	4.14E–02	–	3.24E–01	3.70E–02	–	2.10E–01	2.81E–02	–	2.31E–01	4.95E–02	–	**2.04E–01**	3.03E–02
*f*15	4.47E+00	1.92E+00	–	**1.06E+00**	2.59E–02	+	1.94E+01	1.22E+00	–	7.04E+00	6.56E–01	–	5.27E+00	1.07E+00	–	4.32E+00	1.06E+00
*f*16	1.29E+01	4.82E–01	–	1.30E+01	6.27E–01	–	1.36E+01	1.33E–01	–	1.10E+01	3.15E–01	–	1.14E+01	6.13E–01	–	**9.78E+00**	5.38E–01
*f*17	3.26E+05	2.57E+05	–	3.04E+05	3.72E+04	–	8.00E+06	2.81E+06	–	1.54E+03	2.61E+02	–	2.60E+05	1.66E+05	–	**8.99E+02**	3.06E+02
*f*18	1.02E+06	2.95E+06	≈	5.85E+04	6.43E+04	–	4.71E+04	1.10E+05	–	6.26E+01	1.21E+01	–	1.20E+03	1.48E+03	–	**3.05E+01**	9.87E+00
*f*19	2.82E+02	1.98E+01	–	1.62E+01	3.52E+01	≈	1.91E+01	5.33E–01	–	6.44E+00	6.17E–01	–	1.15E+01	1.04E+01	–	**4.55E+00**	1.14E+00
*f*20	1.08E+04	2.18E+03	–	2.36E+03	3.14E+03	–	1.16E+05	5.49E+04	–	4.85E+01	6.47E+00	–	2.59E+03	2.81E+03	–	**2.47E+01**	5.84E+00
*f*21	7.54E+05	4.21E+05	–	1.34E+05	5.59E+04	–	2.49E+06	7.99E+05	–	6.97E+02	1.41E+02	–	1.69E+05	1.10E+05	–	**4.12E+02**	1.45E+02
*f*22	8.24E+02	5.24E+02	–	1.38E+03	1.28E+02	–	7.63E+02	1.10E+02	–	1.43E+02	4.87E+01	≈	4.02E+02	1.78E+02	–	**1.19E+02**	5.84E+01
*f*23	3.42E+02	6.24E+00	–	3.01E+02	3.87E–02	–	3.37E+02	1.82E+00	–	3.15E+02	1.79E–06	–	3.15E+02	1.69E–03	–	**2.00E+02**	0.00E+00
*f*24	2.05E+02	2.41E–01	–	2.51E+02	1.40E+01	–	2.33E+02	6.53E+00	–	2.25E+02	6.97E–01	–	2.29E+02	5.89E+00	–	**2.00E+02**	0.00E+00
*f*25	2.20E+02	4.51E+00	–	2.18E+02	8.47E+00	–	2.52E+02	1.14E+01	–	2.04E+02	2.95E–01	–	2.13E+02	4.17E+00	–	**2.00E+02**	0.00E+00
*f*26	**1.00E+02**	2.42E–01	≈	1.56E+02	4.44E+01	–	1.01E+02	5.64E–02	≈	**1.00E+02**	5.39E–02	≈	1.10E+02	3.05E+01	≈	1.04E+02	1.82E+01
*f*27	2.51E+03	4.82E+02	–	7.89E+02	2.06E+02	–	1.30E+03	3.82E+01	–	4.01E+02	3.26E–01	–	4.70E+02	8.76E+01	–	**2.27E+02**	6.93E+01
*f*28	1.81E+03	4.91E+02	–	3.46E+03	8.02E+03	–	4.94E+02	2.12E+01	–	9.61E+02	4.21E+01	–	1.31E+03	3.92E+02	–	**2.00E+02**	0.00E+00
*f*29	8.77E+07	3.24E+07	–	1.39E+04	1.97E+05	–	**3.55E+02**	3.46E+01	+	1.01E+03	7.50E+01	–	1.32E+03	3.00E+02	–	6.07E+02	2.14E+02
*f*30	4.11E+05	1.87E+04	–	3.51E+03	2.08E+03	–	1.72E+03	1.64E+02	–	1.12E+03	1.71E+02	≈	2.68E+03	6.47E+02	–	**9.64E+02**	3.95E+02
+	0			2			1			0			3				
–	25			26			28			25			25				
≈	5			2			1			5			2				

Table 4
Comparisons of APSO, OLPSO, DVBA and QS over 30 test functions with 30 dimensions.

Func.	APSO			OLPSO			DVBA			QS	
	MeanErr	StdDev	*t*-test	MeanErr	StdDev	*t*-test	MeanErr	StdDev	*t*-test	MeanErr	StdDev
*f*1	1.38E+05	9.59E+04	–	6.12E+06	3.58E+06	–	1.05E+05	4.28E+05	≈	**3.72E+03**	7.17E+03
*f*2	4.34E–03	1.02E–02	≈	1.28E+03	1.48E+03	–	2.93E+04	1.32E+04	–	**9.76E–04**	7.27E–04
*f*3	2.61E+02	4.10E+02	–	3.23E+02	5.69E+02	–	1.59E+04	6.25E+03	–	**7.90E–05**	3.62E–05
*f*4	6.85E+00	2.03E+01	≈	8.64E+01	2.22E+01	–	1.08E+02	3.31E+01	–	**2.15E–01**	4.34E–01
*f*5	2.00E+01	1.88E–04	+	2.03E+01	1.28E–01	+	2.01E+01	6.11E–02	+	2.04E+01	5.23E–02
*f*6	1.58E+01	3.53E+00	–	5.09E+00	1.48E+00	–	2.34E+01	3.16E+00	–	**4.88E+00**	3.07E+00
*f*7	1.73E–02	2.08E–02	–	**1.02E–13**	3.47E–14	≈	1.07E+00	1.12E–02	–	4.11E–04	2.25E–03
*f*8	8.86E–12	4.67E–11	+	**0.00E+00**	0.00E+00	+	3.00E–02	1.76E–01	+	3.65E–01	6.65E–01
*f*9	9.13E+01	2.46E+01	–	4.06E+01	7.02E+00	+	**2.14E+01**	9.52E+00	+	5.28E+01	1.26E+01
*f*10	7.92E–01	8.25E–01	+	8.72E–02	2.04E–01	+	7.90E+02	1.63E+02	–	2.46E+00	1.90E+00
*f*11	2.74E+03	5.37E+02	–	2.28E+03	4.66E+02	–	3.93E+03	5.23E+02	–	**1.79E+03**	4.78E+02
*f*12	1.95E–01	7.17E–02	+	2.28E–01	6.38E–02	+	9.80E–01	3.07E–01	–	3.93E–01	6.83E–02
*f*13	4.33E–01	9.22E–02	–	**2.59E–01**	3.20E–02	≈	5.10E–01	1.03E–01	–	2.65E–01	5.00E–02
*f*14	3.23E–01	1.10E–01	–	2.41E–01	2.66E–02	–	**2.00E–01**	2.05E–02	≈	2.04E–01	3.03E–02
*f*15	2.96E+01	4.03E+00	–	6.67E+00	1.62E+00	–	**2.49E+00**	6.33E–01	+	4.32E+00	1.06E+00
*f*16	1.05E+01	8.21E–01	–	1.17E+01	5.48E–01	–	1.27E+01	4.27E–01	–	**9.78E+00**	5.38E–01
*f*17	3.19E+04	2.14E+04	–	7.98E+05	4.13E+05	–	8.09E+04	3.42E+04	–	**8.99E+02**	3.06E+02
*f*18	3.73E+03	5.23E+03	–	3.58E+02	5.12E+02	–	7.90E+03	9.35E+03	–	**3.05E+01**	9.87E+00
*f*19	1.41E+01	1.84E+01	–	6.13E+00	8.20E–01	–	1.81E+01	2.33E+00	–	**4.55E+00**	1.14E+00
*f*20	6.38E+03	4.86E+03	–	5.58E+03	4.01E+03	–	8.29E+02	3.22E+02	–	**2.47E+01**	5.84E+00
*f*21	2.16E+04	1.31E+04	–	1.07E+05	8.33E+04	–	8.45E+04	4.28E+04	–	**4.12E+02**	1.45E+02
*f*22	6.50E+02	2.42E+02	–	2.20E+02	1.07E+02	–	4.70E+02	1.76E+02	–	**1.19E+02**	5.84E+01
*f*23	3.15E+02	1.15E–12	–	3.15E+02	1.23E–10	–	3.16E+02	3.61E–01	–	**2.00E+02**	0.00E+00
*f*24	2.29E+02	4.73E+00	–	2.24E+02	5.47E–01	–	2.49E+02	1.04E+01	–	**2.00E+02**	0.00E+00
*f*25	2.16E+02	5.81E+00	–	2.09E+02	1.75E+00	–	2.19E+02	9.11E+00	–	**2.00E+02**	0.00E+00
*f*26	1.58E+02	6.05E+01	–	**1.00E+02**	4.44E–02	≈	**1.00E+02**	7.53E–02	≈	1.04E+02	1.82E+01
*f*27	6.84E+02	2.11E+02	–	3.26E+02	3.80E+01	–	5.70E+02	2.61E+02	–	**2.27E+02**	6.93E+01
*f*28	2.53E+03	8.16E+02	–	8.73E+02	2.97E+01	–	1.56E+03	3.08E+02	–	**2.00E+02**	0.00E+00
*f*29	1.24E+03	5.02E+02	–	1.36E+03	2.82E+02	–	1.84E+04	2.20E+04	–	**6.07E+02**	2.14E+02
*f*30	2.50E+03	6.63E+02	–	2.39E+03	5.99E+02	–	1.19E+04	5.09E+03	–	**9.64E+02**	3.95E+02
+	4			5			4				
–	24			22			23				
≈	2			3			3				

error and standard deviation are calculated over 30 independent runs of each algorithm with MFES = 300,000. Comparison results are listed in Table 3.

As shown in Table 3, for unimodal functions (f_1–f_3), QS shows competitive performance compared with five standard algorithms. For multimodal functions (f_4–f_{16}), QS shows better comprehensive performance than the other algorithms. For f_5, f_{12} and f_{13}, the performance of QS closely follows the best performance of BBO. For f_7, in terms of two-tailed *t*-test, the performances of QS and SFS are relatively close, which are better than those of the other four algorithms. Similarly, for f_8, the performance of QS is close to that of PSO in terms of two-tailed *t*-test, and the other four algorithms have worse performance. For f_9 and f_{15}, the performance of QS is only worse than that of GA. For hybrid functions (f_{17}–f_{22}), QS displays better performance than the other algorithms. For composition functions (f_{23}–f_{30}), QS also presents better comprehensive performance than the other algorithms. For f_{26}, the performance of QS is very close to the best performance of PSO and SFS in terms of two-tailed *t*-test. For f_{29}, only the performance of ABC is better than that of QS. Finally, based on the statistical results of two-tailed *t*-tests, it can be concluded that QS is a highly competitive metaheuristic algorithm compared with the five standard algorithms.

Besides, Fig. 4 shows some typical convergence curves of QS and five standard algorithms for a part of bound-constrained benchmark functions with 30 dimensions from CEC 2014. It can be seen that QS does not show an obvious advantage in early iterations. One reason leading to this phenomenon is that QS distributes individuals to different local research regions based on the FFBIs instead of gathering all the individuals in a local region based on the current best individual. However, this distribution mechanism improves the global search ability of QS. As shown in Fig. 4(d), (l), (o) and (p), QS can effectively jump out of a local optimal solution and has the competitive ability of searching the global optimum.

6.4. Comparison 2: QS and six state-of-the-art modified algorithms

In this section, the comparison of QS and six state-of-the-art modified algorithms is conducted. The mean errors and standard deviations of APSO, OLPSO and DVBA are obtained under 30 independent runs, which are taken from [39] and presented in Table 4. The results of EPSDE, JADE and SaDE are obtained under 51 independent runs, which are taken from

Table 5
Comparisons of EPSDE, JADE, SaDE and QS over 30 test functions with 30 dimensions.

Func.	EPSDE			JADE			SADE			QS	
	MeanErr	StdDev	*t*-test	MeanErr	StdDev	*t*-test	MeanErr	StdDev	*t*-test	MeanErr	StdDev
$f1$	2.4162E+04	8.26E+04	–	**4.4782E+02**	1.04E+03	+	2.9897E+05	2.33E+05	–	2.98E+03	5.90E+03
$f2$	**0.0000E+00**	0.00E+00	+	**0.0000E+00**	0.00E+00	+	**0.0000E+00**	0.00E+00	+	9.07E–04	6.52E–04
$f3$	**0.0000E+00**	0.00E+00	+	5.6292E–04	2.51E–03	≈	1.4258E+01	5.83E+01	≈	7.97E–05	3.88E–05
$f4$	3.2127E+00	2.23E+00	≈	**0.0000E+00**	0.00E+00	≈	3.7184E+01	3.76E+01	–	1.50E+00	9.65E+00
$f5$	2.0347E+01	3.22E–02	+	**2.0287E+01**	3.79E–02	+	2.0536E+01	5.55E–02	–	2.04E+01	5.63E–02
$f6$	1.8893E+01	1.64E+00	–	9.4229E+00	2.16E+00	–	5.4599E+00	2.00E+00	≈	**4.88E+00**	2.66E+00
$f7$	2.0760E–03	5.53E–03	–	**0.0000E+00**	0.00E+00	≈	1.2333E–02	2.00E–02	–	2.41E–04	1.72E–03
$f8$	**0.0000E+00**	0.00E+00	+	**0.0000E+00**	0.00E+00	+	7.8036E–02	3.36E–01	+	3.71E–01	8.66E–01
$f9$	4.4362E+01	6.76E+00	+	**2.6166E+01**	4.18E+00	+	3.8121E+01	9.12E+00	+	5.37E+01	1.22E+01
$f10$	2.0140E–01	2.30E–01	+	**5.3069E–03**	1.01E–02	+	2.6919E–01	4.62E–01	+	2.33E+00	1.67E+00
$f11$	3.5646E+03	3.42E+02	–	**1.6399E+03**	2.64E+02	≈	3.1475E+03	6.97E+02	–	1.77E+03	4.75E+02
$f12$	5.2516E–01	7.99E–02	–	**2.7113E–01**	3.75E–02	+	7.9419E–01	1.14E–01	–	3.90E–01	6.11E–02
$f13$	2.4299E–01	4.35E–02	+	**2.2030E–01**	3.91E–02	+	2.5159E–01	4.04E–02	≈	2.64E–01	5.14E–02
$f14$	2.7812E–01	6.16E–02	–	2.3399E–01	3.00E–02	–	2.2853E–01	3.61E–02	–	**2.08E–01**	3.64E–02
$f15$	5.6693E+00	8.57E–01	–	**3.0980E+00**	4.30E–01	+	4.1410E+00	1.39E+00	≈	4.25E+00	9.35E–01
$f16$	1.1146E+01	3.04E–01	–	**9.3698E+00**	3.64E–01	+	1.0911E+01	2.75E–01	–	9.69E+00	6.20E–01
$f17$	4.6053E+04	5.98E+04	–	9.6734E+03	5.99E+04	≈	1.1531E+04	1.04E+04	–	**8.97E+02**	2.55E+02
$f18$	3.3188E+02	3.97E+02	–	3.5805E+02	1.51E+03	≈	4.4383E+02	6.53E+02	–	**2.91E+01**	8.45E+00
$f19$	1.3300E+01	9.95E–01	–	4.4373E+00	6.70E–01	≈	**4.0013E+00**	8.63E–01	≈	4.36E+00	1.12E+00
$f20$	5.0042E+01	8.14E+01	–	2.8854E+03	2.33E+03	–	1.2454E+02	9.57E+01	–	**2.46E+01**	5.96E+00
$f21$	8.8443E+03	1.20E+04	–	7.5750E+03	3.79E+04	≈	2.8350E+03	3.06E+03	–	**4.20E+02**	1.31E+02
$f22$	2.5561E+02	8.32E+01	–	1.4570E+02	7.25E+01	–	1.6633E+02	5.45E+01	–	**1.14E+02**	6.30E+01
$f23$	3.1401E+02	9.00E–13	–	3.1524E+02	2.30E–13	–	3.1524E+02	5.01E–13	–	**2.00E+02**	0.00E+00
$f24$	2.2884E+02	6.64E+00	–	2.2561E+02	3.59E+00	–	2.2862E+02	5.05E+00	–	**2.00E+02**	0.00E+00
$f25$	2.0023E+02	4.06E–02	–	2.0388E+02	1.27E+00	–	2.0828E+02	2.31E+00	–	**2.00E+02**	0.00E+00
$f26$	**1.0026E+02**	3.94E–02	≈	1.0218E+02	1.40E+01	≈	1.0614E+02	2.37E+01	≈	1.06E+02	2.37E+01
$f27$	8.3334E+02	1.16E+02	–	3.4415E+02	5.16E+01	–	4.1786E+02	3.76E+01	–	**2.16E+02**	5.44E+01
$f28$	3.9831E+02	1.45E+01	–	8.0273E+02	3.58E+01	–	8.9045E+02	2.94E+01	–	**2.00E+02**	0.00E+00
$f29$	**2.1430E+02**	1.14E+00	+	7.5909E+02	1.59E+02	–	1.0792E+03	1.87E+02	–	6.09E+02	2.19E+02
$f30$	**5.6708E+02**	1.12E+02	+	1.9192E+03	6.41E+02	–	1.6680E+03	5.72E+02	–	8.83E+02	3.45E+02
+	9			10			4				
–	19			11			20				
≈	2			9			6				

[40] and listed in Table 5. To make a fair comparison, QS is run 30 times and 51 times for each test function, respectively. For all algorithms, the MFES = 300,000.

As shown in Table 4, for unimodal functions (f_1–f_3), hybrid functions (f_{17}–f_{22}) and composition functions (f_{23}–f_{30}), QS is superior to the other three algorithms except for f_{26}. For f_{26}, the mean error of QS closely follows the smallest one of OLPSO and DVBA, and QS does not show significant difference from them in terms of two-tailed *t*-tests. For multimodal functions (f_4–f_{16}), each of APSO, OLPSO and DVBA has the smallest mean errors for three functions, and QS has the smallest mean errors for the other four functions. According to the statistical results of two-tailed *t*-tests for all functions, QS shows the highly competitive performance compared with APSO, OLPSO and DVBA.

As shown in Table 5, for most of unimodal functions (f_1–f_3) and multimodal functions (f_4–f_{16}), JADE performs better than QS. However, for most of hybrid functions (f_{17}–f_{22}) and composition functions (f_{23}–f_{30}), QS performs better than JADE. For functions f_2, f_3, f_8, f_{26}, f_{29} and f_{30}, EPSDE gains the smallest mean errors. SaDE shows the smallest mean errors for f_2 and f_{19}. According to the statistical results of two-tailed *t*-tests for all functions, it can be seen that QS is also a more competitive algorithm compared with EPSDE, JADE and SaDE.

6.5. Comparison 3: QS and six state-of-the-art algorithms for high-dimensional functions

In this section, the performance of QS on high-dimensional problems is checked by its comparison with six state-of-the-art algorithms. 30 bound-constrained benchmark functions with 100 dimensions from CEC 2014 are chosen and used for the comparison. Six comparison algorithms are LSHADE44 competed in special session at CEC 2016, and FWA-DM, NRGA, b3e3pbest, OPTBees and b636rl competed in special session at CEC 2014. For each test function, comparison results are obtained under 51 independent runs of each algorithm with MFES = 1,000,000. The results of LSHADE44 are taken from [37] and the results of FWA-DM, NRGA, b3e3pbest, OPTBees and b636rl are taken from [28]. Comparison results are listed in Table 6.

As shown in Table 6, for unimodal functions (f_1–f_3), multimodal functions (f_7, f_{11}, f_{13}–f_{15}) and hybrid functions (f_{17}, f_{18}, f_{20}–f_{22}), LSHADE44 performs the best. For multimodal functions (f_6, f_7) and composition functions (f_{23}–f_{25}, f_{27}, f_{28}), QS performs the best. According to the statistical results of two-tailed *t*-tests for all functions, QS is outperformed by LSHADE44.

Table 6
Comparisons of FWA-DM, NRGA, LSHADE44, b3e3pbest, OPTBees, b636rl and QS over 30 test functions with 100 dimensions.

Func.	FWA-DM			NRGA			LSHADE44			b3e3pbest			OPTBees			b636rl			QS	
	MeanErr	StdDev	*t*-test	MeanErr	StdDev	*t*-test	MeanErr	StdDev	*t*-test	MeanErr	StdDev	*t*-test	MeanErr	StdDev	*t*-test	MeanErr	StdDev	*t*-test	MeanErr	StdDev
*f*1	2.28E+08	4.08E+07	–	3.24E+07	4.56E+06	–	**2.84E+05**	8.53E+04	+	4.34E+05	2.07E+05	+	3.00E+05	1.00E+05	+	1.31E+06	4.07E+05	+	2.47E+06	7.31E+05
*f*2	1.62E+04	1.81E+04	–	1.46E+04	6.68E+03	–	**6.29E–05**	2.00E–04	+	1.57E+04	2.09E+04	–	1.09E+01	2.92E+01	–	2.22E+04	2.69E+04	–	2.41E+00	1.98E+00
*f*3	2.95E+04	3.64E+03	–	2.70E+04	4.06E+03	–	**7.21E–09**	1.16E–08	+	9.13E+01	1.43E+02	–	7.50E+02	7.86E+02	–	6.94E+02	7.93E+02	–	9.48E–01	7.57E–01
*f*4	1.83E+02	1.00E+02	–	3.96E+02	3.51E+01	–	1.85E+02	3.06E+01	≈	1.43E+02	3.93E+01	+	**1.42E+02**	5.48E+01	+	1.74E+02	3.34E+01	≈	1.75E+02	3.96E+01
*f*5	2.10E+01	2.44E–02	–	2.00E+01	8.09E–05	+	2.06E+01	2.76E–02	+	2.07E+01	2.16E–02	–	**2.00E+01**	2.49E–05	+	2.06E+01	2.30E–02	+	2.06E+01	3.18E–02
*f*6	1.14E+02	2.54E+00	–	9.76E+01	4.93E+00	–	7.22E+01	2.25E+00	–	6.94E+01	1.05E+01	–	7.15E+01	7.73E+00	–	7.40E+01	2.98E+00	–	**6.01E+01**	7.30E+00
*f*7	1.29E–01	3.14E–02	–	2.22E–02	7.34E–03	–	**0.00E+00**	0.00E+00	≈	6.32E–03	1.23E–02	–	5.89E–03	8.05E–03	–	5.68E–08	1.09E–08	–	**0.00E+00**	0.00E+00
*f*8	1.08E+02	5.40E+00	–	2.00E+02	2.39E+01	–	9.81E–06	4.14E–06	+	**0.00E+00**	1.64E–10	+	1.42E–12	3.05E–13	+	1.08E–08	5.06E–09	+	6.86E+01	1.90E+01
*f*9	5.52E+02	4.44E+01	–	**2.45E+02**	2.22E+01	+	1.18E+02	8.80E+00	+	3.11E+02	4.08E+01	+	6.66E+02	9.79E+01	–	3.21E+02	3.44E+01	+	3.91E+02	5.15E+01
*f*10	5.67E+03	3.04E+02	–	6.33E+03	1.28E+03	–	4.21E+00	8.69E–01	+	**3.67E–03**	4.53E–03	+	4.26E+03	4.30E+02	–	5.30E–02	4.50E–02	+	2.11E+02	1.64E+02
*f*11	1.46E+04	6.30E+02	–	1.37E+04	1.56E+03	–	**1.15E+04**	4.48E+02	≈	1.29E+04	1.37E+03	–	1.24E+04	1.11E+03	–	1.18E+04	4.80E+02	≈	1.17E+04	1.03E+03
*f*12	1.21E+00	7.78E–02	–	3.80E–01	8.91E–02	+	4.61E–01	3.24E–02	≈	6.44E–01	7.49E–02	–	**2.32E–01**	5.08E–02	+	5.04E–01	3.20E–02	≈	4.73E–01	1.88E–01
*f*13	5.61E–01	3.61E–02	–	5.01E–01	3.00E–02	–	**3.27E–01**	2.07E–02	+	4.90E–01	6.04E–02	–	5.90E–01	8.16E–02	–	5.23E–01	5.20E–02	–	4.64E–01	5.15E–02
*f*14	1.89E–01	2.06E–02	+	1.63E–01	7.21E–03	–	**1.17E–01**	1.10E–02	+	2.99E–01	3.06E–02	≈	2.27E–01	2.27E–02	+	2.18E–01	1.60E–02	+	3.06E–01	7.56E–02
*f*15	8.74E+01	5.87E+00	–	4.53E+02	5.23E+01	–	**1.77E+01**	2.63E+00	+	5.68E+01	1.17E+01	–	6.59E+01	1.83E+01	–	4.09E+01	4.47E+00	–	3.38E+01	6.33E+00
*f*16	4.35E+01	3.75E–01	–	4.36E+01	1.04E+00	–	3.98E+01	4.34E–01	≈	4.04E+01	4.63E–01	≈	4.09E+01	1.17E+00	–	**3.97E+01**	6.54E–01	+	4.03E+01	9.78E–01
*f*17	2.31E+07	5.63E+06	–	2.17E+06	4.83E+05	–	**4.49E+03**	6.71E+02	+	4.51E+04	1.81E+04	–	1.09E+05	6.74E+04	–	1.86E+05	5.31E+04	–	3.42E+04	1.45E+04
*f*18	5.68E+03	8.70E+03	–	6.32E+02	3.11E+02	+	**2.29E+02**	1.87E+01	+	2.35E+03	2.13E+03	–	1.59E+03	2.11E+03	–	9.41E+02	8.65E+02	+	1.41E+03	1.83E+03
*f*19	6.34E+01	2.43E+00	+	9.93E+01	1.97E+01	–	9.38E+01	2.32E+00	–	8.65E+01	3.17E+01	–	**5.28E+01**	1.56E+01	+	9.47E+01	1.20E+01	–	8.44E+01	2.84E+01
*f*20	6.93E+04	1.10E+04	–	7.17E+04	1.42E+04	–	**1.42E+02**	3.33E+01	+	9.50E+02	3.42E+02	–	1.06E+04	4.24E+03	–	7.73E+03	3.38E+03	–	2.81E+02	5.19E+01
*f*21	9.57E+06	2.31E+06	–	1.92E+06	4.77E+05	–	**1.91E+03**	5.25E+02	+	2.62E+04	1.66E+04	–	3.11E+05	1.60E+05	–	8.92E+04	3.92E+04	–	1.19E+04	8.73E+03
*f*22	1.51E+03	1.34E+02	–	2.29E+03	4.53E+02	–	**1.42E+03**	2.53E+02	≈	1.60E+03	3.40E+02	–	2.03E+03	3.31E+02	–	1.88E+03	2.31E+02	–	1.42E+03	2.87E+02
*f*23	3.46E+02	2.18E–01	–	3.70E+02	3.30E+00	–	3.48E+02	2.3E–13	–	3.48E+02	2.30E–13	–	3.46E+02	9.28E–01	–	3.48E+02	2.24E–08	–	**2.00E+02**	0.00E+00
*f*24	3.63E+02	2.85E+00	–	3.76E+02	4.32E+00	–	3.93E+02	2.57E+00	–	4.16E+02	8.06E+00	–	3.49E+02	1.05E+01	–	3.63E+02	2.38E+00	–	**2.00E+02**	0.00E+00
*f*25	3.03E+02	1.74E+01	–	2.27E+02	2.07E+01	–	2.15E+02	1.26E+01	–	2.64E+02	1.39E+01	–	2.08E+02	1.12E+00	–	2.48E+02	9.05E+00	–	**2.00E+02**	0.00E+00
*f*26	1.61E+02	5.88E+01	+	2.00E+02	4.18E–02	≈	1.96E+02	1.95E+01	≈	2.00E+02	1.20E–02	≈	**1.01E+02**	6.84E–02	+	2.00E+02	4.40E–02	≈	2.00E+02	0.00E+00
*f*27	3.14E+03	4.13E+02	–	2.35E+03	1.34E+02	–	3.29E+02	2.21E+01	–	1.77E+03	1.68E+02	–	2.16E+03	1.67E+02	–	2.02E+03	1.90E+02	–	**2.00E+02**	0.00E+00
*f*28	1.60E+03	3.42E+02	–	1.21E+04	8.95E+02	–	2.25E+03	5.32E+01	–	2.25E+03	3.79E+02	–	6.14E+02	4.04E+01	–	3.05E+03	7.68E+02	–	**2.00E+02**	0.00E+00
*f*29	**2.70E+02**	5.23E+00	+	3.81E+03	4.67E+02	–	9.04E+02	1.92E+02	+	1.13E+03	4.32E+02	+	2.75E+02	3.27E+00	+	1.61E+03	1.80E+02	+	2.06E+03	5.75E+02
*f*30	**2.23E+03**	1.16E+03	+	8.73E+04	1.92E+04	–	8.38E+03	1.07E+03	≈	1.08E+04	2.01E+03	–	2.86E+03	2.42E+00	+	8.54E+03	1.17E+03	≈	8.66E+03	1.93E+03
+	5			3			15			6			10			9				
–	25			26			7			21			20			16				
≈	0			1			8			3			0			5				

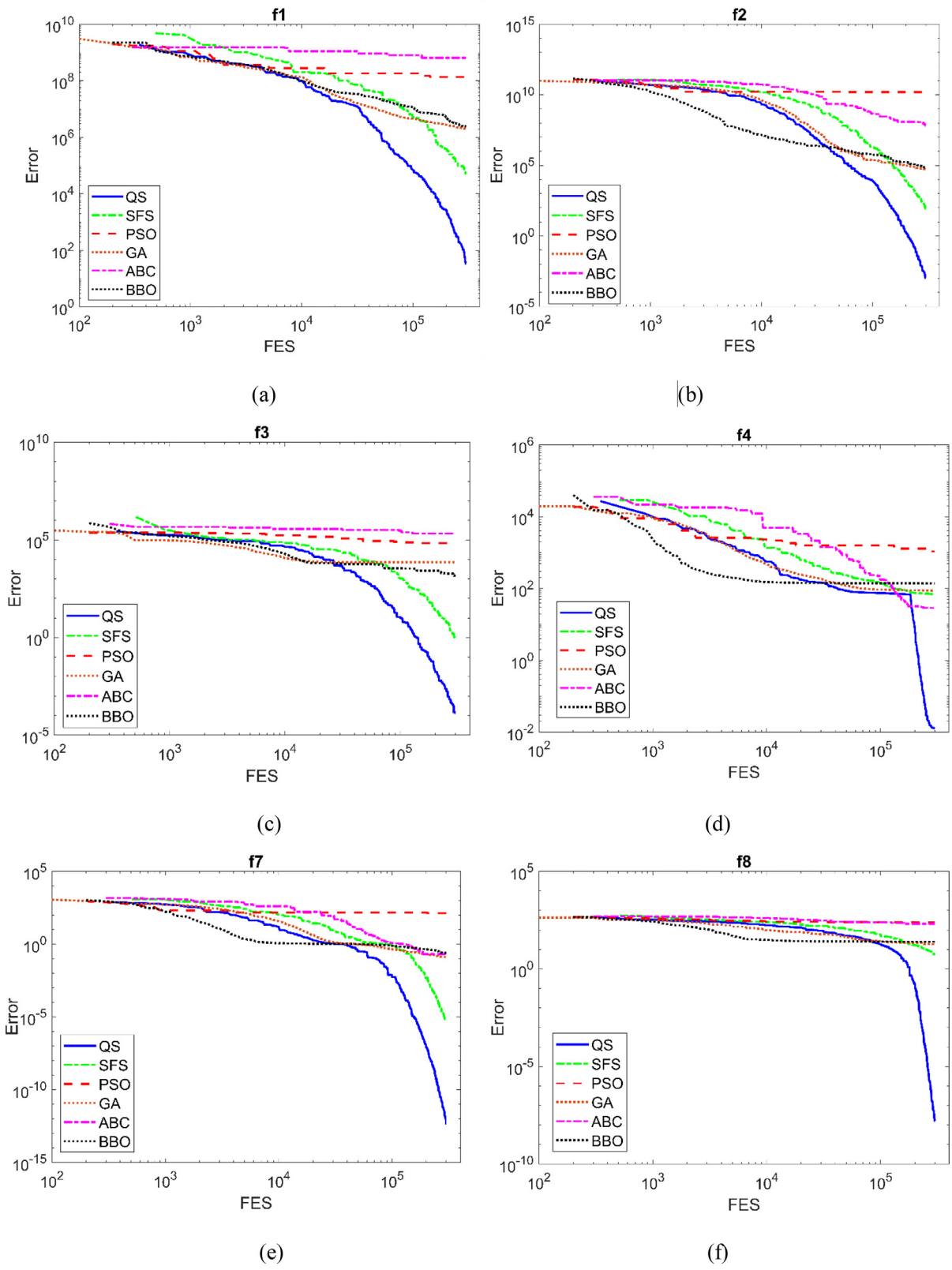

Fig. 4. Some typical convergence curves of QS and five standard algorithms for a part of bound-constrained benchmark functions with 30 dimensions from CEC 2014.

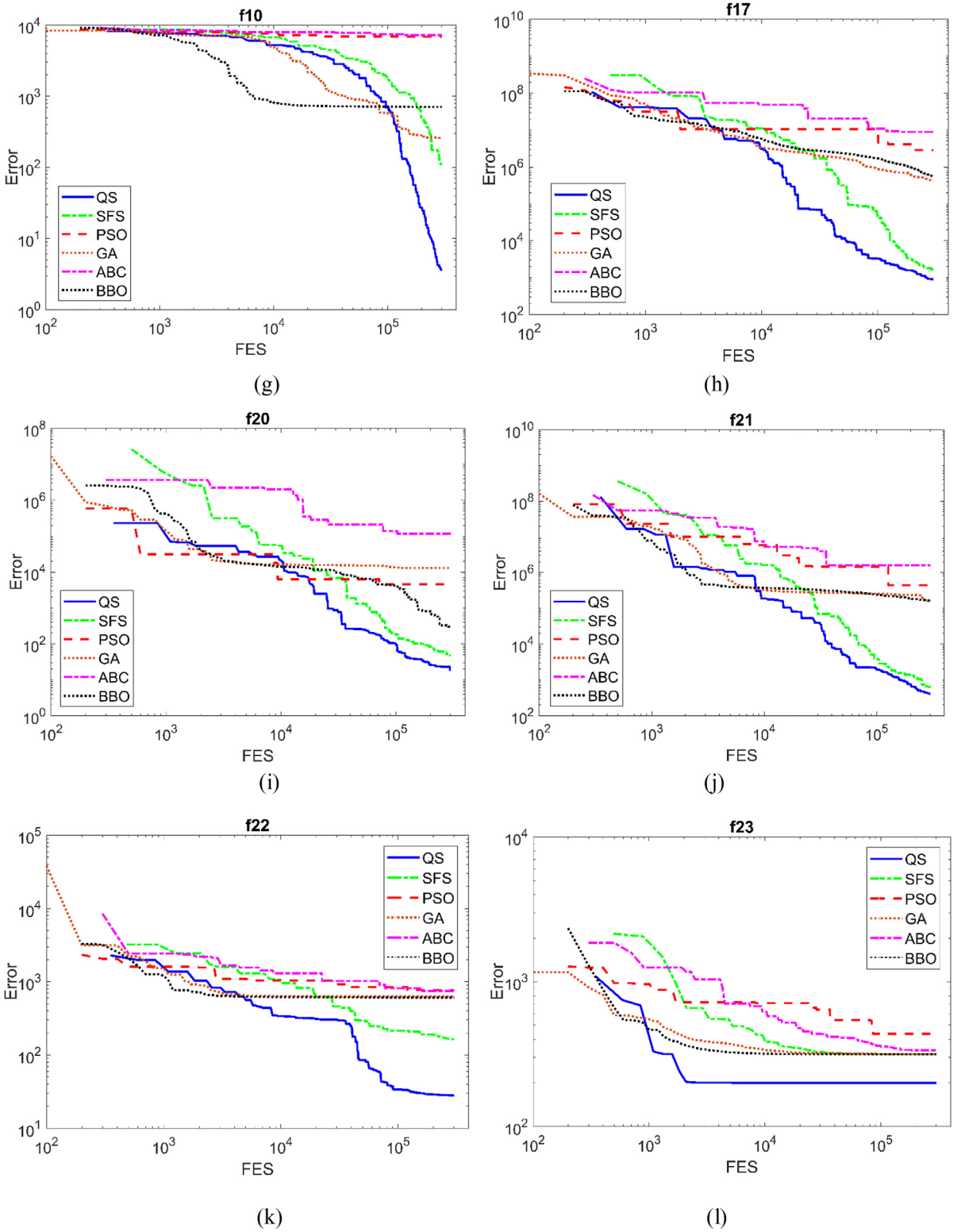

Fig. 4. Continued

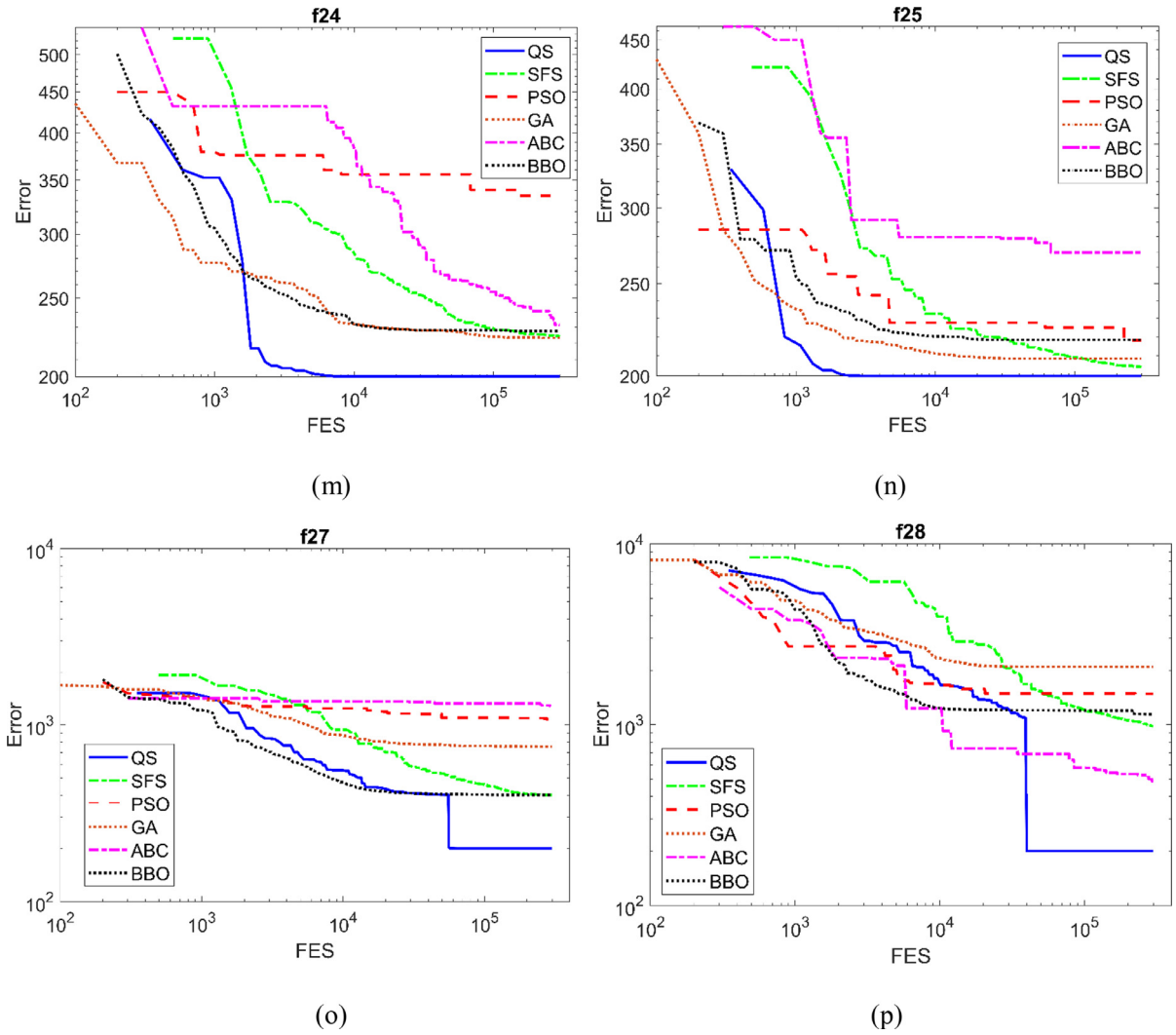

Fig. 4. Continued

Nevertheless, compared with the other five state-of-the-art algorithms, QS shows better performance. Thus, QS is still a competitive algorithm for the high-dimensional optimization problems.

7. Test of constrained engineering optimization problems

In the test of bound-constrained benchmark functions, QS shows the high competitiveness compared with other standard and state-of-the-art metaheuristic algorithms, which lays a solid foundation for the following test of constrained engineering optimization problems. In this section, the performance of QS is tested by five extensively used constrained mechanical design problems and four challenging constrained engineering optimization problems.

7.1. Test of five constrained mechanical design problems

Five constrained mechanical design problems are chosen to check the performance of the QS algorithm in this section, including designs of welded beam, speed reducer, pressure vessel, tension/compression spring, and bearing. The detailed descriptions about these problems can be found in [13,15]. For each problem, the best, mean and worst values of the objective function are obtained for comparisons under 25 independent runs of each algorithm.

7.1.1. Welded beam design optimization

This problem is to minimize the cost of a welded beam and has been solved using different algorithms, such as particle swarm optimization-differential evolution TLBO [13], WCA [15], ABC [41], (PSO-DE) [42], constraint violation with interval

Table 7
Comparison results for the welded beam design optimization.

Algorithms	Best	Mean	Worst	MFES
TLBO	**1.724852**	1.728447	NA	20,000
WCA	1.724856	1.726427	1.744697	46,450
	1.724857	1.73594	1.801127	30,000
ABC	**1.724852**	1.741913	NA	30,000
PSO-DE	**1.724852**	**1.724852**	**1.724852**	66,600
CVI-PSO	**1.724852**	1.725124	1.727665	25,000
BIANCA	**1.724852**	1.752201	1.793233	80,000
BA	1.7312	1.878656	2.345579	20,000
MBA	1.724853	1.724853	1.724853	47,340
FFA	**1.724852**	**1.724852**	**1.724852**	50,000
PVS	**1.724852**	**1.724852**	**1.724852**	50,000
	1.724852	1.724887	1.725056	20,000
QS	**1.724852**	**1.724852**	**1.724852**	20,000

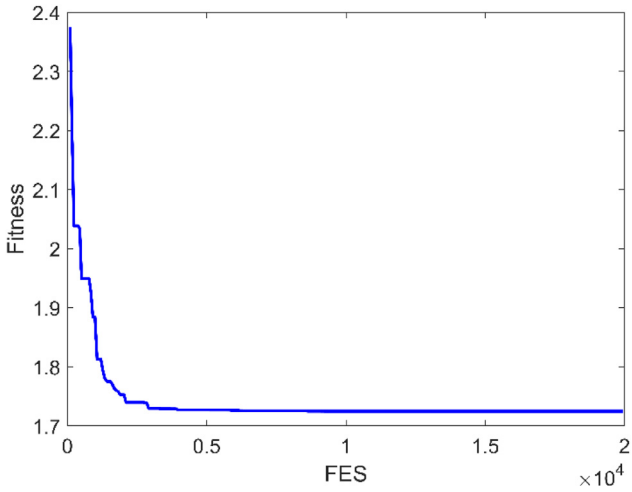

Fig. 5. Convergence process of QS for the welded beam design optimization.

arithmetic PSO (CVI-PSO) [43], multi-population GA (BIANCA) [44], bat algorithm (BA) [45], mine blast algorithm (MBA) [46], firefly algorithm (FFA) [47] and PVS [14]. In QS for this problem, the population size and MFES are set to 30 and 20,000, respectively. Table 7 summarizes the best, mean and worst values of the objective function obtained by all the comparison algorithms. As shown in Table 7, PSO-DE, FFA, PVS and QS obtain the best results compared with the other algorithms. However, 20,000 MFES in QS is obviously less than 66,600 MFES of PSO-DE and 50,000 MFES of FFA and PVS. Fig. 5 illustrates the convergence process of QS for the welded beam design optimization. It can be observed that the optimization efficiency of QS is high since it can rapidly find a good search region within 5000 FES. Therefore, QS performs the best among all the comparison algorithms in terms of solving accuracy and efficiency.

7.1.2. Speed reducer design optimization

(1) Speed reducer design optimization under case 1

This problem is to minimize the weight of a speed reducer. Seven design variables exist: face width (x_1), module of teeth (x_2), number of teeth in the pinion (x_3), length of the first shaft between bearings (x_4), length of the second shaft between bearings (x_5), and the diameters of the first and second shafts (x_6, x_7). In this case, the range of x_5 is [7.8, 8.3]. This problem has been solved by PSO-DE [42], ABC [41], TLBO [13], FFA [47], and PVS [14]. In QS, the population size is set to 30, and MFES is set to 20,000 and 25,000, respectively. Comparison results are shown in Table 8. As shown in Table 8, it can be seen that the best results are obtained by PVS with MFES = 54,350 and QS with MFES = 25,000. QS shows a higher optimization efficiency due to its less MFES. Under MFES = 20,000, QS obtains better results than PVS. Moreover, these results are very close to its best results under MFES = 25,000. The convergence process of QS for the speed reducer design optimization under case 1 is presented in Fig. 6.

(2) Speed reducer design optimization under case 2

This problem is a different version of the speed reducer design, where the length of the second shaft between bearings ranges from 7.3 to 8.3. It has been solved by DELC [48], WCA [15], MBA [46], and PVS [14]. For QS, this problem is solved by considering a population size of 30 with 15,000 and 25,000 MFES, respectively. Comparison results are summarized in Table 9. It can be seen that the best results are obtained by DELC with MFES = 30,000 and QS with MFES = 25,000. QS

Table 8
Comparison results for the speed reducer design optimization under case 1.

Algorithms	Best	Mean	Worst	MFES
PSO-DE	2996.348167	2996.348174	2996.348204	54,350
ABC	2997.058	2997.058	NA	30,000
TLBO	2996.34817	2996.34817	NA	20,000
FFA	2996.37	2996.51	2996.669	50,000
PVS	**2996.348165**	**2996.348165**	**2996.348165**	54,350
	2996.348165	2996.350001	2996.366218	20,000
QS	**2996.348165**	**2996.348165**	**2996.348165**	25,000
	2996.348165	**2996.348165**	2996.348166	20,000

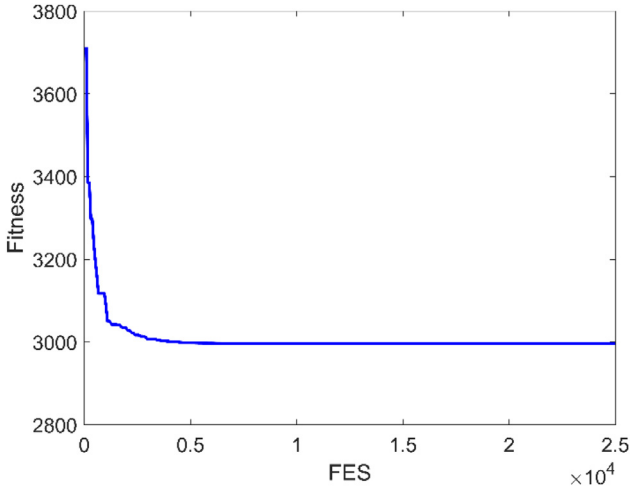

Fig. 6. Convergence process of QS for the speed reducer design optimization under case 1.

Table 9
Comparison results for the speed reducer design optimization under case 2.

Algorithms	Best	Mean	Worst	MFES
DELC	**2994.471066**	**2994.471066**	**2994.471066**	30,000
WCA	**2994.471066**	2994.474392	2994.505578	15,150
MBA	2994.4824	2996.769	2999.65	6300
PVS	**2994.471066**	2994.483258	2994.752395	15,150
	2994.471066	2994.472059	2994.477593	30,000
QS	2994.471090	2994.471129	2994.471242	15,000
	2994.471066	**2994.471066**	**2994.471066**	25,000

displays higher optimization efficiency due to its less MFES. Under MFES = 15,150, the best values of WCA and PVS are the smallest; however, their mean and worst values are larger than those of QS with MFES = 15,000. Fig. 7 illustrates the convergence process of QS for the speed reducer design optimization under case 2.

7.1.3. Pressure vessel design optimization

(1) Pressure vessel design optimization under case 1

There are four design variables in this problem: thickness of the shell x_1, thickness of the head x_2, inner radius x_3, and length of the cylindrical section of the vessel x_4. In this case, x_1 and x_2 are expected to be integer multiples of 0.0625, and x_3 and x_4 are continuous design variables. Some metaheuristic algorithms have been used to solve this problem, including TLBO [13], ABC [41], CVI-PSO [43], BIANCA [44], diversity enhanced constrained PSO (DEC-PSO) [49], BA [45], FFA [47] and PVS [14]. A population size of 50 is taken into account in QS and the stopping criterion is 20,000 MFES. Comparison results are summarized in Table 10. It can be seen that QS is capable of obtaining the best value. The mean value obtained by QS is better than those by the other algorithms except TLBO and DEC-PSO. Moreover, the worst value obtained by QS is not worse than those by other algorithms. Fig. 8 shows the convergence process of QS for the pressure vessel design optimization under case 1.

(2) Pressure vessel design optimization under case 2

This problem is another version of the pressure vessel design, where all the four design variables are continuous. This problem has been solved by Nelder-Mead PSO (NM-PSO) [50], WCA [15], MBA [46], and PVS [14]. For comparison, 20,000

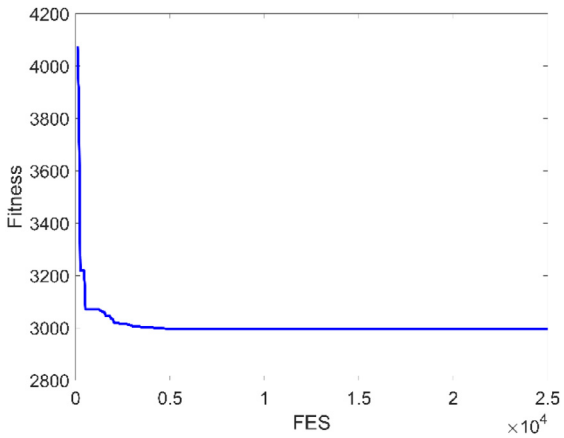

Fig. 7. Convergence process of QS for the speed reducer design optimization under case 2.

Table 10
Comparison results for the pressure vessel design optimization under case 1.

Algorithms	Best	Mean	Worst	MFES
TLBO	**6059.714**	**6059.714**	NA	20,000
ABC	**6059.714**	6245.308	NA	30,000
CVI-PSO	**6059.714**	6292.123	6820.41	25,000
BIANCA	6059.938	6182.002	6447.325	80,000
DEC-PSO	**6059.714**	6060.33	**6090.526**	300,000
BA	**6059.714**	6179.13	6318.95	20,000
FFA	**6059.714**	6064.33	**6090.52**	50,000
PVS	**6059.714**	6065.877	**6090.526**	20,000
QS	**6059.714**	6060.947	**6090.526**	20,000

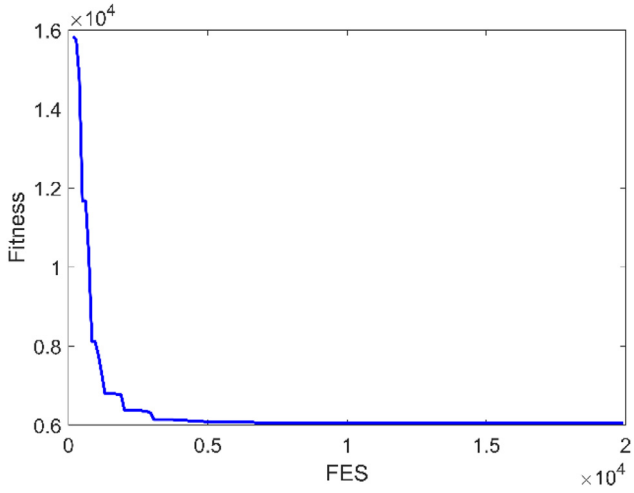

Fig. 8. Convergence process of QS for the pressure vessel design optimization under case 1.

MFES and a population size of 30 are used for QS. Comparison results are listed in Table 11. It can be observed that QS has the best performance in terms of the best, mean and worst values. Fig. 9 shows the convergence process of QS for the pressure vessel design optimization under case 2.

7.1.4. Tension/compression spring design optimization

This problem is to minimize the weight of the spring under four constraints. In this problem, QS is compared with 12 different algorithms: PSO-DE [42], TLBO [13], ABC [41], WCA [15], CVI-PSO [43], BIANCA [44], BA [45], interior search algorithm (ISA) [51], FFA [47] and PVS [14]. A population size of 50 is set for QS and comparison results are shown in Table 12. As shown in Table 12, it can be seen that only BIANCA cannot find the optimum value. QS obtains the best performance in terms of the best, mean and worst values with 40,000 MFES, which is less than the 42,100 MFES of PSO-DE

Table 11
Comparison results for the pressure vessel design optimization under case 2.

Algorithms	Best	Mean	Worst	MFES
NM-PSO	5930.314	5946.79	5960.056	80,000
WCA	**5885.333**	6198.617	6590.213	27,500
MBA	5889.321	6200.64	6392.5	70,650
PVS	**5885.333**	5885.409	5886.035	20,000
QS	**5885.333**	**5885.333**	**5885.336**	20,000

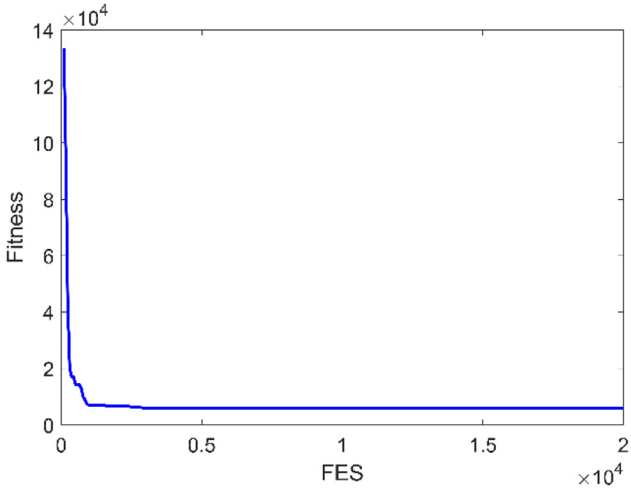

Fig. 9. Convergence process of QS for the pressure vessel design optimization under case 2.

Table 12
Comparison results of the spring design optimization.

Algorithms	Best	Mean	Worst	MFES
PSO-DE	**0.012665**	**0.012665**	**0.012665**	42,100
TLBO	**0.012665**	0.012666	NA	20,000
ABC	**0.012665**	0.012709	NA	30,000
WCA	**0.012665**	0.012746	0.012952	11,750
CVI-PSO	**0.012666**	0.012731	0.012843	25,000
BIANCA	0.012671	0.012681	0.012913	80,000
BA	**0.012665**	0.013501	0.016895	20,000
ISA	**0.012665**	0.012799	0.013165	8000
FFA	**0.012665**	0.012677	0.000013	50,000
PVS	**0.012665**	**0.012665**	**0.012665**	42,100
QS	**0.012665**	**0.012665**	**0.012665**	40,000
	0.012665	0.012666	0.012669	20,000
	0.012665	0.012691	0.012828	8000

and PVS. In addition, under 20,000 MFES, QS obtains the same results as TLBO, but outperforms ABC, CVI-PSO, BIANCA, BA, and FFA in terms of the mean values. Under 8,000 MFES, QS outperforms WCA and ISA in terms of the mean values. Fig. 10 shows the convergence process of QS for the spring design optimization.

7.1.5. Bearing design optimization

This problem is to maximize the dynamic load carrying capacity with 9 constraints and 10 design variables. The number of bearing balls has to take an integer value. This problem has been solved by some algorithms: GA4 [52], ABC [41], TLBO [13], multi-member diversity-based DE (MDDE) [53] and PVS [14]. For QS, a population size of 30 and 10,000 MFES are adopted. Comparison results are tabulated in Table 13. It can be noticed that QS outperforms all the other algorithms. Fig. 11 gives the convergence process of QS for the bearing design optimization.

7.2. Test of 4 challenging constrained engineering optimization problems

To further test the performance of the QS algorithm, 4 challenging constrained engineering optimization problems are chosen in this section, including designs of multi-plate disc clutch brake, Belleville spring, planetary gear train and stiffened

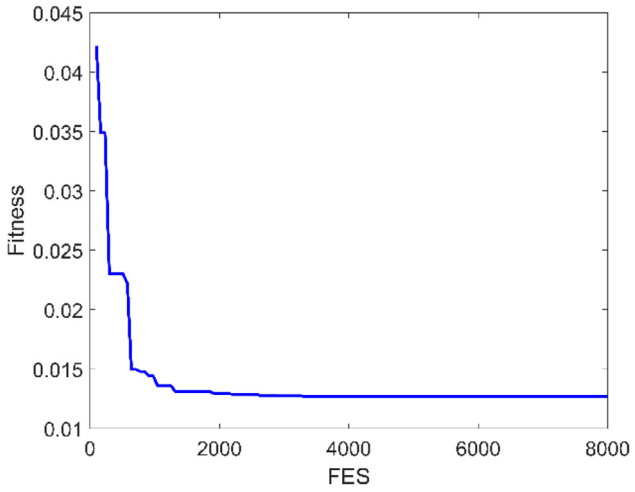

Fig. 10. Convergence process of QS for the spring design optimization.

Table 13
Comparison results for the bearing design optimization.

Algorithms	Best	Mean	Worst	MFES
GA4	81843.3	NA	NA	225,000
ABC	**81859.74**	81496	78897.81	10,000
TLBO	**81859.74**	81438.98	80807.85	20,000
MDDE	81858.83	81848.7	81701.18	10,000
PVS	81859.59	80803.57	78897.81	10,000
	81859.74	81550	79834.79	20,000
QS	**81859.74**	**81859.74**	**81859.74**	10,000

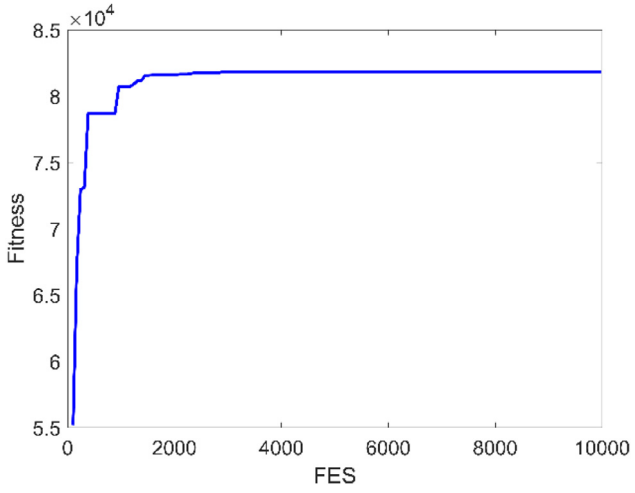

Fig. 11. Convergence process of QS for the bearing design optimization.

welded shell. The detailed descriptions about these problems can be found in [13,14]. For each problem, the best, mean and worst values of the objective function are obtained for comparisons under 25 independent runs of each algorithm.

7.2.1. Multi-plate disc clutch brake design optimization

This problem is to minimize the weight of a multi-plate disc clutch brake by considering 5 discrete design variables. The results obtained by QS are compared with those by ABC [41], TLBO [13], WCA [15], and PVS [14]. QS is implemented using a population size of 30 with 600 and 1200 MFES, respectively. All results are shown in Table 14. It can be found that all the algorithms can obtain the same best value. In terms of the mean and worst values, QS is inferior to WCA, but better than ABC, TLBO and PVS. Fig. 12 illustrates the convergence process of QS for the multi-plate disc clutch brake design optimization.

Table 14
Comparison results for the multi-plate disc clutch brake design optimization.

Algorithms	Best	Mean	Worst	MFES
ABC	**0.313657**	0.324751	0.352864	600
TLBO	**0.313657**	0.327166	0.392071	1200
WCA	**0.313657**	**0.313657**	**0.313657**	500
PVS	**0.313657**	0.333652	0.352864	600
	0.313657	0.328163	0.392071	1200
QS	**0.313657**	0.322082	0.343748	600
	0.313657	0.313970	0.317577	1200

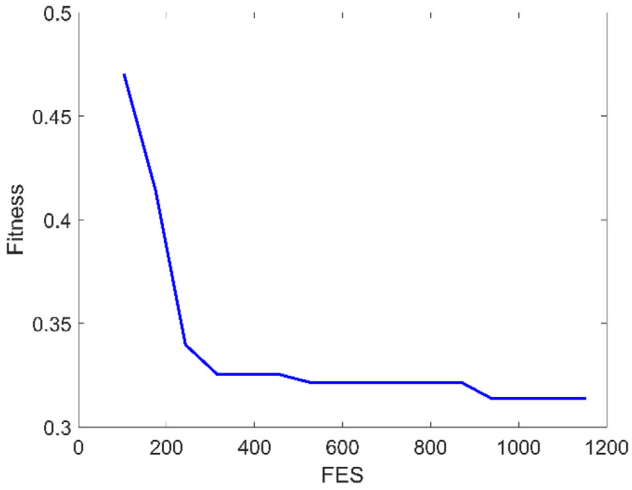

Fig. 12. Convergence process of QS for the multi-plate disc clutch brake design optimization.

Table 15
Comparison results for the Belleville spring optimization problem.

Algorithms	Best	Mean	Worst	MFES
TLBO	**1.979675**	1.979687	1.979757	15,000
ABC	**1.979675**	1.995475	2.104297	15,000
MDDE	**1.979675**	1.982256	2.104326	15,000
BBO	2.081124	2.229694	NA	15,000
GSA	2.400689	2.666064	NA	15,000
FPA	1.984885	1.990434	NA	15,000
PVS	1.979688	1.983524	NA	15,000
QS	**1.979676**	**1.979684**	**1.979741**	15,000

7.2.2. Belleville spring design optimization

The problem is required to minimize the weight of Belleville spring with 4 design variables: thickness, height, external and internal diameters of the spring. The thickness is a discrete variable whereas others are continuous variables. It has been addressed using various algorithms: TLBO [13], ABC [41], MDDE [53], BBO [14], GSA [14], FPA [14] and PVS [14]. In QS, a population size of 30 and 15,000 MFES are applied. Comparison results are tabulated in Table 15. It can be obviously noticed that QS outperforms all the other algorithms. Fig. 13 shows the convergence process of QS for the Belleville spring design optimization.

7.2.3. Planetary gear train design optimization

This problem is to synthesize the gear-teeth number for an automatic planetary transmission system which is used in automobiles to minimize the maximum errors in the gear ratio [14]. Six design variables are considered based on the number of teeth in the gears, which can only take integer values. There are 3 discrete design variables and 11 constraints, one of which is an equality constraint. The ratio of the feasible region relative to the total search space is less than 0.0001 [14]. This problem is solved by QS with 25,000 MFES and a population size of 30. QS is compared with ABC-GA [54], ABC-DE [54], ABC-BBO [54], TLBO [54], BBO [14], GSA [14], FPA [14] and PVS [14]. Comparison results are summarized in Table 16. As shown in Table 16, QS outperformed all the other algorithms in terms of the mean value. The best value of QS is inferior to that of FPA, very close to that of PVS, and better than those of the other algorithms. For this problem, worst values are

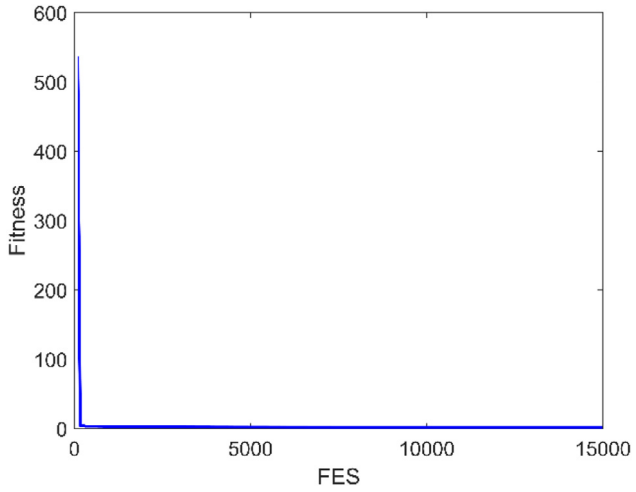

Fig. 13. Convergence process of QS for the Belleville spring design optimization.

Table 16
Comparison results for the planetary gear train design optimization.

Algorithm	Best	Mean	MFES
ABC-GA	0.53	0.53669	25,000
ABC-DE	0.527814	0.54479	25,000
ABC-BBO	0.52735	0.52908	25,000
TLBO	0.52735	0.53371	25,000
BBO	0.532222	0.57247	25,000
GSA	0.53623	0.73245	25,000
FPA	**0.52325**	0.52978	25,000
PVS	0.525588	0.53063	25,000
QS	0.525589	**0.528393**	25,000

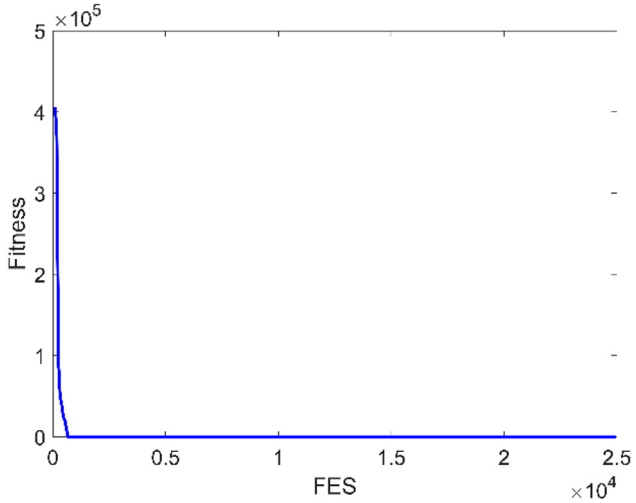

Fig. 14. Convergence process of QS for the planetary gear train design optimization.

not presented in references, so only the best and mean values are selected for comparisons. Fig. 14 shows the convergence process of QS for the planetary gear train design optimization.

7.2.4. Stiffened welded shell design optimization

This problem requires the minimization of the cost for a cylindrical shell member, which is orthogonally stiffened using ring stiffeners and stringers with a halved I-section. Five design variables which only have specified discrete values are involved: shell thickness, number of longitudinal stiffeners, number of ring stiffeners, box height, and stringer stiffness height. In QS, 15,000 MFES and a population size of 30 are used to solve this problem. QS is compared with ABC-GA [54],

Table 17
Comparison results for stiffened welded shell design optimization.

Algorithms	Best	Mean	MFES
ABC-GA	55724.831	56920.4	15,000
ABC-DE	55326.293	55852.8	15,000
ABC-BBO	55326.293	55921.9	15,000
TLBO	55326.293	55667.4	15,000
BBO	55724.831	56783.8	15,000
GSA	55944.771	56746	15,000
FPA	55326.293	55590.9	15,000
PVS	55326.293	55531.2	15,000
QS	**53415.321**	**53509.3**	15,000

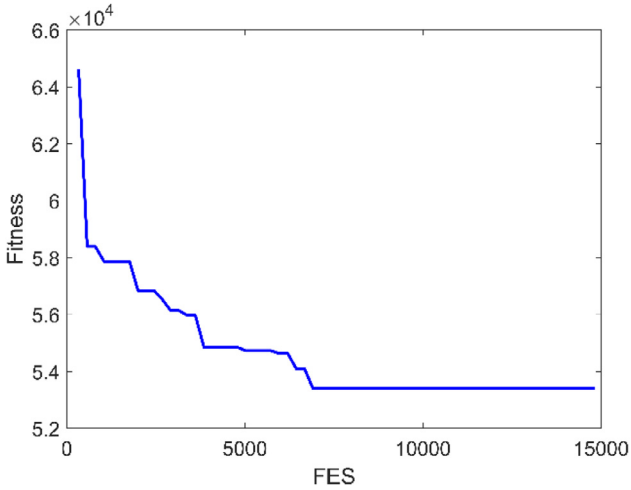

Fig. 15. Convergence process of QS for the stiffened welded shell design optimization.

ABC-DE [54], ABC-BBO [54], TLBO [54], BBO [14], GSA [14], FPA [14] and PVS [14]. Comparison results are summarized in Table 17. According to these results, it can be seen that QS performs the best among all the algorithms. Fig. 15 shows the convergence process of QS for the stiffened welded shell design optimization.

8. Conclusions

From the inspiration of human activities in queuing process, a novel metaheuristic algorithm, queuing search (QS) algorithm, is proposed in this paper. Unlike most of existing metaheuristic algorithms, QS does not need to preset the other parameters except the population size and stopping criterion, which are two common parameters in all metaheuristic algorithms. This feature makes QS can be easily applied to new optimization problems.

Performance of QS is checked by solving 30 bound-constrained benchmark function optimization problems with 30 and 100 dimensions from CEC 2014, 5 standard and 4 challenging constrained engineering optimization problems. After its comparison with some state-of-the-art or well-known metaheuristic algorithms, experiment results show that the performance of QS is outstanding. The superiority and competitiveness of QS on the ability of jumping out of a local optimum and searching the global optimum are demonstrated.

As for metaheuristic algorithms, both the simplicity and ability of searching the global optimum are crucial for their applications in real-life engineering optimization problems. The proposed QS algorithm can be easily applied to new optimization problems without adjusting the other parameters except the population size and stopping criterion. Furthermore, it has the strong competitive ability of searching the global optimum compared to many other metaheuristic algorithms. Therefore, these advantages of the proposed QS algorithm make it as a good candidate for solving the solution of real-life engineering optimization problems.

When fitness functions in some optimization problems involve the output of the time-consuming or expensive-to-evaluate black-box functions, e.g., finite element models, the proposed QS algorithms can be implemented with assistance of metamodels, such as neural network and radial basis functions, which are used to replace the original fitness functions. In the future, the efficient combination of QS and metamodels can be investigated.

Acknowledgment

This research was supported by the National Natural Science Foundation of China [grant numbers 51675196 and 51421062].

References

[1] M. Khajehzadeh, M.R. Taha, A. El-Shafie, M. Eslami, A survey on meta-heuristic global optimization algorithms, Res. J. Appl. Sci. Eng. Technol. 3 (2011) 569–578.
[2] D. Karaboga, B. Akay, A comparative study of artificial bee colony algorithm, Appl. Math. Comput. 214 (2009) 108–132.
[3] J.H. Holland, An Introduction with Application to Biology, Control and Artificial Intelligence Adaptation in Natural and Artificial System, MIT Press, 1975.
[4] J.H. Holland, Adaptation in Natural and Artificial Systems, Univ. of Mich. Press, 1975.
[5] R. Storn, K. Price, Differential evolution – a simple and efficient heuristic for global optimization over continuous spaces, J. Global Optim. 11 (1997) 341–359.
[6] V. Kennedy, R. Eberhart, Particle swarm optimization, in: Proceedings of the IEEE International Conference on Neural Networks, 1995, pp. 1942–1948.
[7] D. Karaboga, An Idea Based on Honey Bee Swarm for Numerical Optimization, Erciyes University, Engineering Faculty, Computer Engineering Department, 2005 Technical Report TR06.
[8] D. Simon, Biogeography-based optimization, IEEE Trans. Evolut. Comput. 12 (2008) 702–713.
[9] A. Askarzadeh, A novel metaheuristic method for solving constrained engineering optimization problems: crow search algorithm, Comput. Struct. 169 (2016) 1–12.
[10] A.R. Mehrabian, C. Lucas, A novel numerical optimization algorithm inspired from weed colonization, Ecol. Inf. 1 (2006) 355–366.
[11] X.S. Yang, Flower pollination algorithm for global optimization, Unconventional Computation and Natural Computation, Springer, Berlin, 2012, pp. 240–249.
[12] Z.W. Geem, J.H. Kim, G.V. Loganathan, A new heuristic optimization algorithm: harmony search, Simulation 76 (2001) 60–68.
[13] R.V. Rao, V.J. Savsani, D.P. Vakharia, Teaching-learning-based optimization: a novel method for constrained mechanical design optimization problems, Comput. Aided Des. 43 (2011) 303–315.
[14] P. Savsani, V. Savsani, Passing vehicle search (PVS): a novel metaheuristic algorithm, Appl. Math. Model. 40 (2016) 3951–3978.
[15] H. Eskandar, A. Sadollah, A. Bahreininejad, M. Hamdi, Water cycle algorithm: a novel meta-heuristic optimization for solving constrained engineering optimization problems, Comput. Struct. 110–111 (2012) 151–166.
[16] H. Salimi, Stochastic fractal search: a powerful metaheuristic algorithm, Knowl. Based Syst. 75 (2015) 1–18.
[17] H. Tao, J.M. Zain, M.M. Ahmed, A.N. Abdalla, W. Jing, A wavelet-based particle swarm optimization algorithm for digital image watermarking, Integr. Comput. Aid. E. 19 (2012) 81–91.
[18] M.M. Joly, T. Verstraete, G. Paniagua, Integrated multifidelity, multidisciplinary evolutionary design optimization of counterrotating compressors, Integr. Comput. Aid. E. 21 (2014) 249–261.
[19] H. Karimi, M. Setak, A bi-objective incomplete hub location-routing problem with flow shipment scheduling, Appl. Math. Model. 57 (2018) 406–431.
[20] J. Pei, X. Liu, B. Liao, P.M. Pardalos, M. Kong, Single-machine scheduling with learning effect and resource-dependent processing times in the serial–batching production, Appl. Math. Model. 58 (2017) 245–253.
[21] R. Rudek, The single machine total weighted completion time scheduling problem with the sum-of-processing time based models: strongly NP-hard, Appl. Math. Model. 50 (2017) 314–332.
[22] Q.K. Pan, T. Meng, B. Wang, L. Gao, A mathematical model and two-stage heuristic for hot rolling scheduling in compact strip production, Appl. Math. Model. 48 (2017) 516–533.
[23] D.H. Wolpert, W.G. Macready, No free lunch theorems for optimization, IEEE Trans. Evolut. Comput. 1 (1) (1997) 67–82.
[24] J. Liang, B. Qu, P.N. Suganthan, Problem Definitions and Evaluation Criteria for the CEC 2014 Special Session and Competition on Single Objective Real-Parameter Numerical Optimization, Computational Intelligence Laboratory, Zhengzhou University, Zhengzhou China and Technical Report, Nanyang Technological University, Singapore, 2013.
[25] S.S. Rao, Engineering Optimization: Theory and Practice, New Age International, New Delhi, 2002.
[26] Z.H. Zhan, J. Zhang, Y. Li, H.S.-H. Chung, Adaptive particle swarm optimization, IEEE Trans. Syst., Man, Cybern. Part B Cybern. 39 (2009) 1362–1381.
[27] Z.H. Zhan, J. Zhang, Y. Li, Y.H. Shi, Orthogonal learning particle swarm optimization, IEEE Trans. Evol. Comput. 15 (2011) 832–847.
[28] A.O. Topal, O. Altun, A novel meta-heuristic algorithm: dynamic virtual bats algorithm, Inform. Sci. 354 (2016) 222–235.
[29] J. Zhang, A.C. Sanderson, An approximate Gaussian model of differential evolution with spherical fitness functions, in: Proceedings of the IEEE Congress on Evolutionary Computing, Singapore, 2007, pp. 2220–2228.
[30] J. Zhang, A.C. Sanderson, JADE: adaptive differential evolution with optional external archive, IEEE Trans. Evol. Comput. 13 (2009) 945–958.
[31] A.K. Qin, P.N. Suganthan, Self-adaptive differential evolution algorithm for numerical optimization, in: Proceedings of the IEEE Congress on Evolutionary Computation, 2005, pp. 1785–1791.
[32] C. Yu, L. Kelley, S. Zheng, Y. Tan, Fireworks algorithm with differential mutation for solving the CEC 2014 competition problems, in: Proceedings of the IEEE Congress on Evolutionary Computation (CEC), IEEE, 2014, pp. 3238–3245.
[33] D. Yashesh, K. Deb, S. Bandaru, Non-uniform mapping in real-coded genetic algorithms, in: Proceedings of the IEEE Congress on Evolutionary Computation (CEC), IEEE, 2014, pp. 2237–2244.
[34] P. Bujok, J. Tvrdik, R. Polakova, Differential evolution with rotation-invariant mutation and competing-strategies adaptation, in: Proceedings of the IEEE Congress on Evolutionary Computation (CEC), IEEE, 2014, pp. 2253–2258.
[35] R.D. Maia, L.N. de Castro, W.M. Caminhas, Real-parameter optimization with optbees,, in: Proceedings of the IEEE Congress on Evolutionary Computation (CEC), IEEE, 2014, pp. 2649–2655.
[36] R. Polakova, J. Tvrdik, P. Bujok, Controlled restart in differential evolution applied to CEC2014 benchmark functions, in: Proceedings of the IEEE Congress on Evolutionary Computation (CEC), IEEE, 2014, pp. 2230–2236.
[37] R. Poláková, J. Tvrdík, P. Bujok, Evaluating the performance of L-Shade with competing strategies on CEC2014 single parameter-operator test suite, in: Proceedings of the 2016 IEEE Congress on Evolutionary Computation (CEC), IEEE, 2016, pp. 1181–1187.
[38] S. Das, A. Abraham, U.K. Chakraborty, A. Konar, Differential evolution using a neighborhood-based mutation operator, IEEE Trans. Evol. Comput. 13 (3) (2009) 526–553.
[39] Y.F. Li, Z.H. Zhan, Y. Lin, J. Zhang, Comparisons study of APSO OLPSO and CLPSO on CEC2005 and CEC2014 test suits, in: Proceedings of the 2015 IEEE Congress on Evolutionary Computation (CEC), IEEE, 2015, pp. 3179–3185.
[40] M.Z. Ali, N.H. Awad, P.N. Suganthan, R.G. Reynolds, An adaptive multipopulation differential evolution with dynamic population reduction, IEEE Trans. Cybern. 47 (2017) 2768–2779.
[41] B. Akay, D. Karaboga, Artificial bee colony algorithm for large-scale problems and engineering design optimization, J. Intell. Manuf. 23 (4) (2012) 1001–1014.
[42] 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.

[43] M. Montemurro, A. Vincenti, P. Vannucci, The automatic dynamic penalisation method (ADP) for handling constraints with genetic algorithms, Comput. Methods Appl. Mech. Eng. 256 (2013) 70–87.
[44] I. Mazhoud, K. Hadj-Hamou, J. Bigeon, P. Joyeux, Particle swarm optimization for solving engineering problems: a new constraint-handling mechanism, Eng. Appl. Artif. Intell. 26 (4) (2013) 1263–1273.
[45] A.H. Gandomi, X.S. Yang, A.H. Alavi, S. Talatahari, Bat algorithm for constrained optimization tasks, Neural Comput. Appl. 22 (6) (2013) 1239–1255.
[46] S. Ali, B. Ardeshir, E. Hadi, H. Mohd, Mine blast algorithm: A new population based algorithm for solving constrained engineering optimization problems, Appl. Soft Comput. 13 (2013) 2592–2612.
[47] B. Adil, B.O. Fehmi, Adaptive firefly algorithm with chaos for mechanical design optimization problems, Appl. Soft Comput. 36 (2015) 152–164.
[48] L. Wang, L.P. Li, An effective differential evolution with level comparison for constrained engineering design, Struct. Multidiscip. Optim. 41 (2010) 947–963.
[49] S. Chun, Y.T. Kim, T.H. Kim, A diversity-enhanced constrained particle swarm optimizer for mixed integer-discrete-continuous engineering design problems, Adv. Mech. Eng. 5 (2013) 130750. http://dx.doi.org/10.1155/2013/130750.
[50] E. Zahara, Y.T. Kao, Hybrid Nelder–Mead simplex search and particle swarm optimization for constrained engineering design problems, Expert Syst. Appl. 36 (2009) 3880–3886.
[51] A.H. Gandomi, Interior search algorithm (ISA): a novel approach for global optimization, ISA Trans. 53 (2014) 1168–1183.
[52] S. Gupta, R. Tiwari, B.N. Shivashankar, Multi-objective design optimization of rolling bearings using genetic algorithm, Mech. Mach. Theory 42 (2007) 1418–1443.
[53] G. Wenyin, C. Zhihua, L. Dingwen, Engineering optimization by means of an improved constrained differential evolution, Comput. Methods Appl. Mech. Eng. 268 (2014) 884–904.
[54] R.V. Rao, V.J. Savsani, Mechanical Design Optimization Using Advanced Optimization Techniques, Springer, London, 2012 ISBN 978-1-4471-2747-5.