



Monkey King Evolution: A new memetic evolutionary algorithm and its application in vehicle fuel consumption optimization



Zhenyu Meng^{a,*}, Jeng-Shyang Pan^{a,b}

^a Department of Computer Science, Harbin Institute of Technology Shenzhen Graduate School, Shenzhen, China

^b College of Information Science and Engineering, Fujian University of Technology, Fuzhou, China

ARTICLE INFO

Article history:

Received 14 July 2015

Revised 7 December 2015

Accepted 8 January 2016

Available online 20 January 2016

Keywords:

Benchmark function

Fuel consumption

Monkey King Evolutionary Algorithm

Number of function evaluation

Particle swarm variants

Vehicle navigation

ABSTRACT

Optimization algorithms are proposed to tackle different complex problems in different areas. In this paper, we firstly put forward a new memetic evolutionary algorithm, named Monkey King Evolutionary (MKE) Algorithm, for global optimization. Then we make a deep analysis of three update schemes for the proposed algorithm. Finally we give an application of this algorithm to solve least gasoline consumption optimization (find the least gasoline consumption path) for vehicle navigation. Although there are many simple and applicable optimization algorithms, such as particle swarm optimization variants (including the canonical PSO, Inertia Weighted PSO, Constriction Coefficients PSO, Fully Informed Particle Swarm, Comprehensive Learning Particle Swarm Optimization, Dynamic Neighborhood Learning Particle Swarm). These algorithms are less powerful than the proposed algorithm in this paper. 28 benchmark functions from BBOB2009 and CEC2013 are used for the validation of robustness and accuracy. Comparison results show that our algorithm outperforms particle swarm optimizer variants not only on robustness and optimization accuracy, but also on convergence speed. Benchmark functions of CEC2008 for large scale optimization are also used to test the large scale optimization characteristic of the proposed algorithm, and it also outperforms others. Finally, we use this algorithm to find the least gasoline consumption path in vehicle navigation, and conducted experiments show that the proposed algorithm outperforms A* algorithm and Dijkstra algorithm as well.

© 2016 Elsevier B.V. All rights reserved.

1. Introduction

Optimization algorithms in evolutionary computation are equipped with a meta-heuristic or stochastic optimization or memetic optimization character, and they belong to the family of trial and error problem solvers and distinguished by the use of a population of candidate solutions. Particles of these algorithms have two main character components, one is exploitation, and the other exploration. Particle Swarm Optimization (PSO) is a powerful evolutionary computational algorithm introduced by Kennedy and Eberhart in [1]. The canonical PSO does not use cross over and mutation operations, and particles in the population produce the next generation by learning from their history best and global best of the population experience. The moving velocity is used to make a balance between the exploitation and exploration of a particle.

As the PSO algorithm is simple, easy to implement and it also has been empirically performed well on many optimization

problems since its inception, many researches have learned about the technique and proposed many variants, or new versions of PSO. [2] proposed a new optimizer using particle swarm theory, and examined how the changes in the paradigm affected the number of iteration required to meet an error criterion. [3] presented a modified particle swarm optimizer with an inertia weight of particle velocity, a 2-dimension 4000-iteration conducted experiment showed that the smaller inertia weight made it converged fast if PSO could find the global optimum. When the inertia weight was small, PSO paid more attention on exploitation and when the inertia weight was larger, PSO paid more on exploration. Moderate value of weight made PSO had the best chance to find the global optimum with a moderate number of iteration. Empirically, inertia weight was set as a decreasing function of iteration instead of a fixed constant. Eberhart and Shi [4] made a comparison between inertia weights and constriction factors in particle swarm optimization, and the experiments showed that constriction coefficient $k = 0.7298$ and the constant $c_1 = c_2 = 2.05$ was a good choice [5]. PSO trajectories and topologies had been deeply analyzed for the importance of the convergence. Kennedy [6,7] claimed that PSO with a small neighborhood might perform better on complex

* Corresponding author. Tel.: +86 13825100875.

E-mail address: mzy1314@gmail.com, mzy_1314@126.com (Z. Meng).

problems, while PSO with a large neighborhood would perform better on simple problems. Suganthan [8] proposed a neighbor operator particle swarm optimization, and the operator calculated the particles distance and particles learned from the neighborhood. Mendes et al. [9] proposed a fully informed particle swarm, valuable information was gained from the particle neighbor and the convergence speed improved by this variant of particle swarm optimization. Mendes [10] gave a deep analysis on population topologies and their influence in particle swarm performance in his doctoral thesis. Other variants such as dynamic multi-swarm optimizer [11] and optimizer proposed in [12,13] performed better to solve shifted-rotated benchmark functions, multi-optimization problems and multi-modal functions respectively. There are also all kinds of applications using PSO to tackle different tasks in different fields. Ref. [14] is an application of optimization traffic lights program with PSO, [15] shows an application to tackle complex network clustering by multi-objective discrete particle swarm optimization, and [16] proposed a binary particle swarm optimization algorithm for optimizing the echo state network.

There are more and more optimization algorithms proposed to tackle specific tough problems, and PSO is a popular one with developing variants from its inception in 1995. PSO variants proposed in recent years for specific problem optimization are much more complicated with huge time consumption than the canonical one, but the optimization results are not so satisfying either on optimization accuracy or on convergence speed. A property that is appealing than just being able to convergence to the optimum when elapsed time approaches infinity, is to guarantee that a good solution can be found with a low number of function evaluations [10]. Simple optimization algorithms with powerful capacity and robust [17] are much popular both for academic researches and engineers. So in this paper, we proposed the MKE algorithm, which has a better convergence speed and convergence accuracy with the similar time complexity in comparison with variants of PSO.

With the development of industry technology, there are more and more cars driving on the road. Traffic navigation becomes a hot topic for city governors and researchers. Different approaches have been advanced to tackle congestion and traffic emergency, which aim for better performance of the traffic networks. The desires of different roles in the traffic networks are different. City governors often emphasize on the output of the total networks, while some of the single drivers pay more attention to least travel time or travel distance and most of them pay attention to least fuel expense. As we know that large fuel consumption occurs when cars are in traffic jams, and how to make a navigation while avoiding congestion in the traffic networks not only achieve good output of the total traffic networks, but also save the drivers' money, and this is an optimization problem. Ref. [18] shows some evolutionary thoughts to tackle vehicle routing problem using a different model of a static network. In this paper we advance models and fitness functions of the traveling fuel consumption for a path in vehicle navigation. The conducted experiments show that our method outperforms A* [19] and Dijkstra [20] in finding the least fuel consumption path of the real-time navigation. The main contributions of the paper include:

1. A new memetic evolutionary algorithm is advanced for global optimization and it outperforms state-of-the-art PSO variants not only on the robustness and accuracy but also on the convergence speed (Test on BBoB2009 [21] and CEC2013 [22] benchmark functions on real parameter optimization).
2. The proposed algorithm has a large scale optimization property that can be well used to tackle large scale optimization problems and it can be easily paralleled on distributed computing systems to boost the calculation speed (Test on CEC2008 [23] benchmark functions on large scale optimization).

3. A traffic networks model based on wireless sensor network environment is proposed with gasoline consumption function (of navigation paths) proposed to be optimized for individual navigation with regarding to least congestion in restricted traffic networks.
4. The navigation result of the proposed algorithm outperforms A* and Dijkstra algorithm on gasoline consumption for real-time navigation.

The rest of the paper is organized as follows, Section 2 presents the related works. Section 3 presents the detailed algorithm of Monkey King Evolution. Section 4 presents the navigation model and fuel consumption fitness function. Section 5 gives a comparative view and analysis and Section 6 shows the final conclusion.

2. Related works

Considerable developments have occurred since the inception of canonical PSO [1]. The canonical PSO is based on swarm intelligence and was inspired by the seeking food behavior of a flock of birds. Individual bird is only influenced by its historical best and the global best of the population, and the evolution equation is shown in Eq. (1). The canonical PSO is simple and easy to implement, but the convergence is not good enough or even rather bad to complicated problems.

$$\begin{cases} V_i^{t+1} \leftarrow V_i^t + c_1 * (X_{fb}^t - X_i^t) + c_2 * (X_{gb}^t - X_i^t); \\ X_i^{t+1} \leftarrow X_i^t + V_i^{t+1}; \end{cases} \quad (1)$$

In order to accelerate the convergence speed of the canonical PSO, an inertial weighted PSO [3] was proposed with the evolution equation shown in Eq. (2). Almost all the PSO variants like constraint coefficient PSO (Eq. (3)), FIPS (Eq. (4)), Comprehensive Learning PSO [12] (CLPSO, learning form personal best and others' best), Cooperative PSO [24] (CPSO, decomposing dimension vector as multiple swarm), Dynamic Neighborhood Learning PSO [13] (dynamic neighborhood topology enabled exploration) use particle topology/relationship for evolution, we can also get the topology perception from Eqs. (1)–(4).

$$\begin{cases} v_i^{t+1} \leftarrow \omega * v_i^t + c_1 * (X_{fb}^t - X_i^t) + c_2 * (X_{gb}^t - X_i^t); \\ X_i^{t+1} \leftarrow X_i^t + v_i^{t+1}; \end{cases} \quad (2)$$

$$\begin{cases} v_i^{t+1} \leftarrow \chi * (v_i^t + \varphi_1 * (X_{fb} - X_i^t) + \varphi_2 * (X_{gb} - X_i^t)); \\ X_i^{t+1} \leftarrow X_i^t + v_i^{t+1}; \end{cases} \quad (3)$$

$$\begin{cases} v(t+1) = \chi * (v(t) + \varphi * (p_m - x(t))) \\ x(t+1) = x(t) + v(t+1) \\ \varphi_k = U[0, \frac{\varphi_{max}}{|N|}], \forall k \in N \\ p_m = \frac{\sum_{k \in N} \omega(k) \varphi_k \otimes p_m}{\sum_{k \in N} \omega(k) \varphi_k} \end{cases} \quad (4)$$

Topology/relationship plays a very important role in the performance of a PSO variant, and proposed topologies up to date still do not make the full exploration of the search region. Some of the algorithms (CPSO, SLPSO) mentioned above need extra computation expense. For example, the computation time complexity of CPSO is about D (D is the dimension number) times larger than PSO, IW-PSO, CCPSO, FIPS, CLPSO, and DNLPSO, and the performance of it does not improved significantly. Moreover, the PSO variants also have a fatal weakness that the performance does not improve with the increase of population size, so is the weakness of these algorithms for parallel computing.

For traffic navigation, Wireless Sensor Networks (WSNs) consisting a number of sensor nodes are used to monitoring a local area and getting the traffic information with little infrastructure [25]. There are often two different types of the structure, one

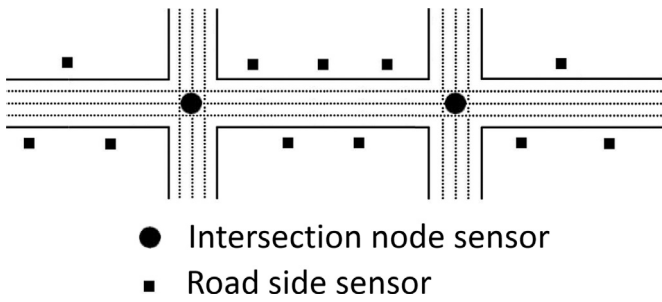


Fig. 1. Four-lane road with sensors in the urban traffic networks.

is unstructured and the other is structured one. The unstructured ones are that contains a dense collection of sensor nodes, and the nodes may be deployed in an ad-hoc manner. For the structured ones, the nodes are deployed in pre-planned manners. The structured style with few nodes can be deployed for traffic information gathering with nodes placed at specific locations to provide coverage. Fig. 1 shows the four-lane road with sensors in the urban traffic network. The WSN communicates with a local area network or wide area network through a gate way. The road-side sensor acts as the end device to collect the traffic information and the intersection node sensor acts as a coordinator and transfers the collected traffic information to the gateway. The gateway acts as a bridge connecting the WSN and other networks, and it enables the data to be processed or stored by other resources. The traffic model used in this paper is simulated on SUMO platform and grid networks. Congestion analysis and least fuel navigation are based on the traffic information collected form the WSNs of urban area.

3. The memetic Monkey King Evolutionary Algorithm

The Monkey King Evolutionary(MKE) Algorithm proposed in this paper is inspired by the action of the Monkey King, a character of a Chinese famous mythological novel, named Journey to the West. The novel relates the amazing adventures of the priest Sanzang as travels west in search of Buddhist Sutras with his three disciples, and Monkey King is the most capable disciple. The journey of the four is supposed to be dangerous and will-stealing. When Monkey King is in trouble, he can transform into different small monkeys to deal with the tough problem, and each small monkey can give a feedback solution to the Monkey King, so the Monkey King can select the best solution for the trouble. The Monkey King algorithm is an updated and more powerful version of Ebb-Tide-Fish algorithm [26]. It is the same as Ebb-Tide-Fish algorithm that there are also only a small amount of particles labeled as Monkey King particles in Monkey King Evolutionary Algorithm. We use a population rate R to define the proportion of particles in the population labeled as Monkey King particles. The Monkey King labels are randomly initialized with the sum equaling to $R * PopSize$, $PopSize$ denotes the population size. In the evolutionary algorithm, each Monkey King particle transforms to a small group of monkeys for exploitation while other particles in the population are for exploration. After each Monkey King particle's exploitation, the particle becomes a normal particle of the population, and then we randomly select $R * PopSize$ particles from the population to change their labels to be new Monkey King particles.

In the MKE algorithm, the number of small monkeys that a Monkey King particle transforms to is $C * D$, C is a constant value while D is the number of dimensions. A larger C value means the Monkey King Particle does more exploitation of a local area and it performs better on multi-modal functions, but it usually increases the computational complexity. Empirically $C = 3$ is a good choice for lower dimension optimizations. The searching behaviors of par-

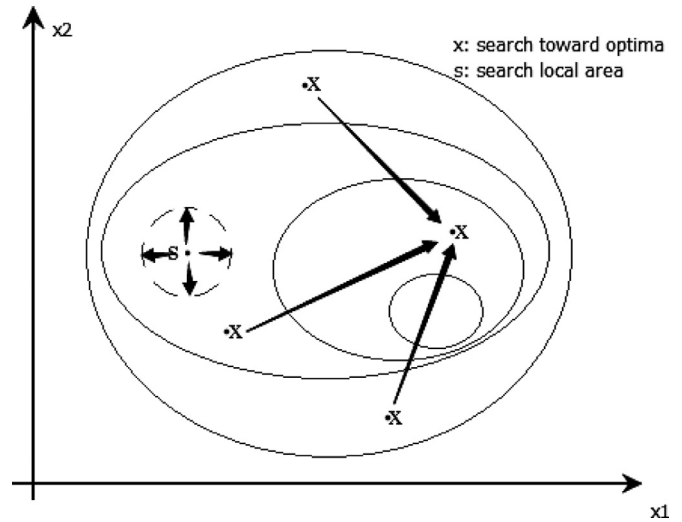


Fig. 2. The search behaviors of particles in Monkey King Evolutionary.

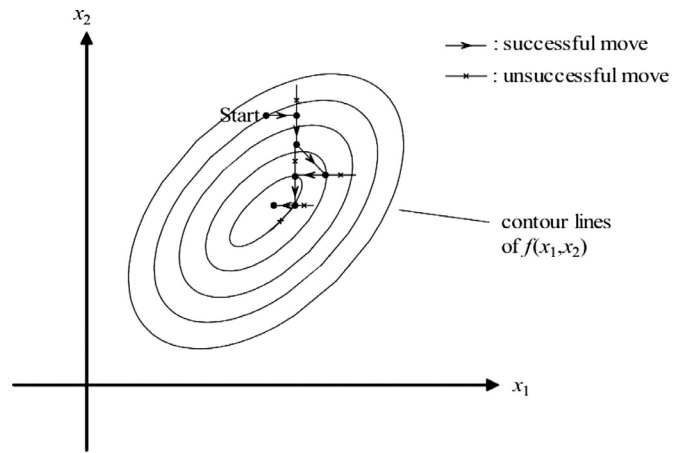


Fig. 3. The motion trajectory of a particle's historical best drawn by global best of the population.

ticles in the population are shown in Fig. 2, different particle labels mean different search styles. The labeled S and X particles in Fig. 2 are denoted as Monkey King particle and common particle respectively. The evolution equation of Monkey King particles and common particles are listed in Eqs. (5), (6) and (7) respectively. This kind of update scheme is denoted as Monkey King Evolution version 1 (MKE_v1). $X_{sm}(i)$ in Eq. (6) denotes the i th “small monkey” particle of the $C \times D$ small monkeys group. All these “small monkey” particles have the same values as $X_{MK, G}$ (a G th-generation Monkey King particle). The “small monkey” particles search the vicinity range of $X_{MK, G}$ by following evolution equation Eq. (5), and $X_{MK, G}$ is updated by $X_{MK, G+1}$, the selected optimum value from the $C \times D$ “small monkey” particles. For the normal particle, evolution follows Eq. (7). $X_{k, pbest}$ denotes the historical best of k th particle in the population, F is the fluctuation coefficient of direction vector (the vector from current position to the global best position). A particle's historical best motion trajectory is shown in Fig. 3, and the pseudo-code of the algorithm is shown below in Algorithm 1.¹

$$X_{sm}(i) = \{x_1, x_2, \dots, x_j, \dots, x_D\}, \tag{5}$$

$$x_j \mapsto x_j \pm 0.2 * rand() * x_j, j \in D.$$

¹ $R(r_1, r_2, \dots, r_d)$ is the bound restraints. As the search domain is usually symmetric, we only need the maximum value of each dimension for the bound restraints, and r_d is the maximum value of d -th dimension.

Algorithm 1 Pseudo-code of the Monkey King Evolution Algorithm.

Initialization:

Initialize the searching space $R(r_1, r_2, \dots, r_d)$ and the benchmark function $f(X)$.

Iteration:

```

1: while exeTime < MaxIteration do
2:   if exeTime = 1 then
3:     Generate the population coordinates  $X_i = (x_{i,1}, x_{i,2}, \dots, x_{i,d})^T$  and generate the Monkey King particle and change its flag.
4:   end if
5:   if exeTime > 1 then
6:     for pSize = 1 : PopSize do
7:       if labeli == 1 then
8:         Monkey king particle evolution (e.g. Eqs. (5) and (6)).
9:         labeli = 0
10:      else
11:        Common particle evolution (e.g. Eq. (7)).
12:      end if
13:    end for
14:    Generate Monkey King and change its flag.
15:  end if
16:  Calculate the fitness value and update  $X_{pbest}$ .
17:  Update the optima with coordinate  $X_{gbest}$ .
18: end while
    
```

Output:

The global optima X_{gbest} and $f(X_{gbest})$.

Table 1

The probability of newly added number of Monkey King particles after one of 50 Monkey King particles finishing its exploitation.

Added MK number	2	1	0
Probability	$\frac{C_{49}^2}{C_{100}^2}$	$\frac{C_{49}^1 \times C_{51}^1}{C_{100}^2}$	$\frac{C_{51}^2}{C_{100}^2}$

$$X_{MK,G+1} = \underset{i \in C \times D}{opt} \{X_{sm}(1), \dots, X_{sm}(i), \dots, X_{sm}(C \times D)\}. \quad (6)$$

$$X_{k,G+1} = X_{k,pbest} + F * rand() * (X_{gbest} - X_{k,G}) \quad (7)$$

Monkey King particles of the population are used as perturbation factors to achieve better optimization results within less computational time, so the proportion rate R is very small. We examine two cases: one is $R * PopSize = 1$, which means there is only one Monkey King particle in the population. The other is $R * PopSize > 1$, we take $R * PopSize = 2$ for analysis. In this case $R * PopSize = 2$, we take $PopSize = 100$ for example, the initial number of Monkey King particles is 2. According to the algorithm, when one Monkey King particle changes its label to normal particle, two other particles in the population are selected and labeled as Monkey King. The number of Monkey King particles is increasing until a certain threshold. We suppose there are half Monkey King particles and half normal particles in the population with the number each 50 particles. When one of the Monkey King particle finishes exploitation and changes its label to normal, two other particles are selected to change their normal labels to be Monkey King labels. Table 1 shows the probability of the number of new added Monkey King particles, so the expectation is 0.74. The balance number n of Monkey King particles satisfies Eq. (8), so $n = 63$. Fig. 4 shows the number of Monkey King in the population with respect to $R * PopSize = 1$ and $R * PopSize = 2$.

$$2 \times \frac{C_{100-n}^2}{C_{100}^2} + \frac{C_n^1 \times C_{100-n}^1}{C_{100}^2} = 0.5 \quad (8)$$

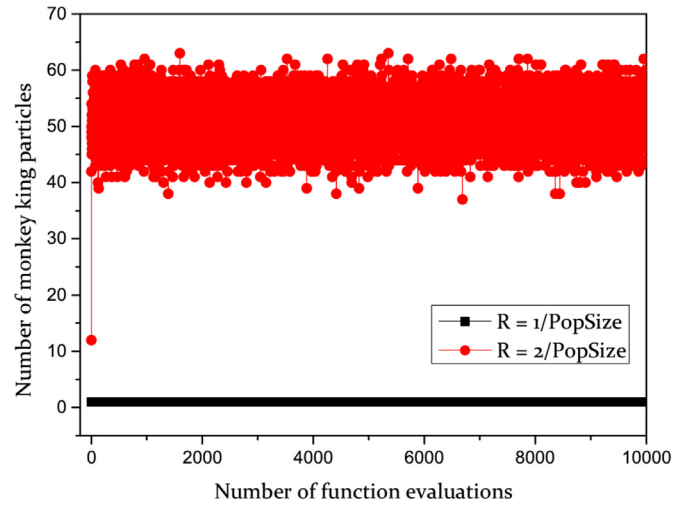


Fig. 4. Number of Monkey King particles of each iteration in 100 particles' population.

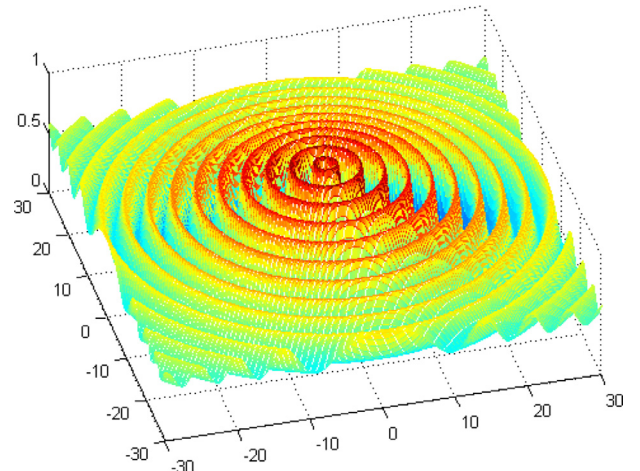


Fig. 5. 2-D schaffer function.

3.1. Benchmark functions

There are many benchmark functions for the validation of new algorithms, these benchmark functions are usually uni-modal functions, multi-modal functions, separable functions, nonseparable functions, symmetrical functions, asymmetrical functions, and even composition functions. Fig. 5 shows a 2-dimension multi-modal schaffer function. In this paper, we use 28 benchmark functions, including uni-modal function, multi-modal function, separable functions, nonseparable functions, symmetry functions and asymmetry functions. The equations of the functions and the minimum values in search domain R of the functions are given in Tables 2 and 3, respectively. The detailed description of these benchmark functions can be found in BB0B2009 noiseless function definitions [21] and CEC2013 problem definition [22].

We also use CEC2008 benchmark functions for the validation of the scale invariant property of the proposed algorithm, and the good large scale optimization property makes the algorithm outperform others on large-scale optimization especially for the application, fuel optimization, in this paper. The optima of CEC2008 benchmark functions are listed in Table 4.

All the experiments are conducted on a PC with Intel(R) Core(TM)2 Duo CPU T6670@ 2.2 Hz on RedHat Linux Enterprise Edition 5.5 Operating System, and all the algorithms are

Table 2
Benchmark functions.

No.	Name	Benchmark function
1	Sphere function	$f_1(x) = \sum_{i=1}^D z_i^2 + f_1^*, Z = X - O$
2	Rot-Hi-Con-Elliptic function	$f_2(x) = \sum_{i=1}^D (10^6)^{\frac{i-1}{D-1}} z_i^2 + f_2^*, Z = T_{osz}(M_1(X - O))$
3	Rot-Be-Cigar function	$f_3(x) = z_1^2 + 10^6 \sum_{i=2}^D z_i^2 + f_3^*, Z = M_2 T_{asy}^{0.5}(M_1(X - O))$
4	Rot-Discuss function	$f_4(x) = 10^6 z_1^2 + \sum_{i=2}^D z_i^2 + f_4^*, Z = T_{osz}(M_1(X - O))$
5	Dif-Powers function	$f_5(x) = \sqrt{\sum_{i=1}^D z_i ^{2+4\frac{i-1}{D-1}}} + f_5^*, Z = X - O$
6	Rot-Ros function	$f_6(x) = \sum_{i=1}^{D-1} (100(z_i^2 - z_{i+1})^2 + (z_i - 1)^2) + f_6^*, Z = M_1(\frac{2.048(X-O)}{100}) + 1$
7	Rot-Schaffers F7 function	$f_7(x) = (\frac{1}{D-1} \sum_{i=1}^{D-1} (\sqrt{z_i} + \sqrt{z_i \sin^2(50z_i^{0.2})}))^2 + f_7^*$ $z_i = \sqrt{y_i^2 + y_{i+1}^2}, Y = \Lambda^{10} M_2 T_{asy}^{0.5}(M_1(X - O))$
8	Rot-Ackley's function	$f_8(x) = -20 \exp(-0.2 \sqrt{\frac{1}{D} \sum_{i=1}^D z_i^2}) - \exp(\frac{1}{D} \sum_{i=1}^D D \cos(2\pi z_i)) + 20 + e + f_8^*$ $Z = \Lambda^{10} M_2 T_{asy}^{0.5}(M_1(X - O))$
9	Rot-weierstrass function	$f_9(x) = \sum_{i=1}^D \sum_{k=0}^{kmax} [a^k \cos(2\pi b^k(z_i + 0.5))] - D \sum_{k=0}^{kmax} [a^k \cos(2\pi b^k 0.5)] + f_9^*$ $a = 0.5, b = 3, kmax = 20, Z = \Lambda^{10} M_2 T_{asy}^{0.5}(M_1(\frac{0.5(X-O)}{100}))$
10	Rot-Griewank's function	$f_{10}(x) = \sum_{i=1}^D \frac{z_i^2}{4000} - \prod_{i=1}^D \cos(\frac{z_i}{\sqrt{i}}) + 1 + f_{10}^*$ $Z = \Lambda^{100} M_1(\frac{600(X-O)}{100})$
11	Rastrigin's function	$f_{11}(x) = \sum_{i=1}^D (z_i^2 - 10 \cos(2\pi z_i) + 10) + f_{11}^*$ $Z = \Lambda^{10} T_{asy}^{0.2}(T_{osz}(\frac{5.12(X-O)}{100}))$
12	Rot-Rastrigin's function	$f_{12}(x) = \sum_{i=1}^D (z_i^2 - 10 \cos(2\pi z_i) + 10) + f_{12}^*$ $Z = M_1 \Lambda^{10} M_2 T_{asy}^{0.2}(T_{osz}(M_1(\frac{5.12(X-O)}{100})))$
13	Non-Rot-Rastrigin's function	$f_{13}(x) = \sum_{i=1}^D (z_i^2 - 10 \cos(2\pi z_i) + 10) + f_{13}^*, Z = M_1 \Lambda^{10} M_2 T_{asy}^{0.2}(T_{osz}(Y))$ $\hat{x} = M_1 \frac{5.12(X-O)}{100}, y_i = \begin{cases} \hat{x}_i, & \text{if } \hat{x}_i \leq 0.5 \\ \text{round}(2\hat{x}_i)/2, & \text{if } \hat{x}_i > 0.5 \end{cases}$
14	Schwefel's function	$f_{14}(Z) = 418.9829 * D - \sum_{i=1}^D g_i(z_i) + f_{14}^*$ $Z = \Lambda^{10} (\frac{100(X-O)}{100}) + 4.209687462275036e + 002$
15	Rot-Schwefel's function	$f_{15}(Z) = 418.9829 * D - \sum_{i=1}^D g_i(z_i) + f_{15}^*$ $Z = \Lambda^{10} M_1(\frac{100(X-O)}{100}) + 4.209687462275036e + 002$
16	Rot-Katsuura function	$f_{16}(x) = \frac{10}{D} \prod_{i=1}^D (1 + i \sum_{j=1}^i z_j - \text{round}(2z_j/2))^{\frac{10}{D-1}} - \frac{10}{D} + f_{16}^*$ $Z = M_2 \Lambda^{100} (M_1(\frac{5(X-O)}{100}))$
17	Lun-Bi-Rastrigin function	$f_{17}(x) = \min(\sum_{i=1}^D y_0^2, dD + s \sum_{i=1}^D y_i^2) + 10(D - \sum_{i=1}^D \cos(2\pi \hat{z}_i)) + f_{17}^*$ $y_0 = (\hat{x}_i - \mu_0), y_1 = (\hat{x}_i - \mu_1), Z = \Lambda^{100}(\hat{x} - \mu_0)$
18	Rot-Lun-Bi-Rastrigin function	$f_{18}(x) = \min(\sum_{i=1}^D y_0^2, dD + s \sum_{i=1}^D y_i^2) + 10(D - \sum_{i=1}^D \cos(2\pi \hat{z}_i)) + f_{18}^*$ $y_0 = (\hat{x}_i - \mu_0), y_1 = (\hat{x}_i - \mu_1), Z = M_2 \Lambda^{100}(M_1(\hat{x} - \mu_0))$
19	Rot-Exp-Gri-plus-Rosenbrock's function	$f_{19}(x) = g_1(g_2(z_1, z_2)) + g_1(g_2(z_2, z_3)) + \dots + g_1(g_2(z_D, z_1)) + f_{19}^*$ $g_1(x) = \sum_{i=1}^D \frac{x_i^2}{4000} - \prod_{i=1}^D \cos(\frac{x_i}{\sqrt{i}}) + 1, z = M_1(\frac{5(X-O)}{100}) + 1$
20	Rot-Exp-Scaffer's F6 function	$f_{20}(x) = g(z_1, z_2) + g(z_2, z_3) + \dots + g(z_D, z_1) + f_{20}^*$ $g(x, y) = 0.5 + \frac{\sin^2(\sqrt{x^2+y^2}-0.5)}{[1+0.001(x^2+y^2)]^2}, Z = M_2 T_{asy}^{0.5}(M_1(X - O))$
21	Composition function 1	$f(x) = \sum_{i=1}^n \omega_i * [\lambda_i g_i(x) + bias_i] + f^*$ $f_i^* = f_i - f_i^*, g_i = f_6^*, g_2 = f_5^*, g_3 = f_3^*, g_4 = f_4^*, g_5 = f_1^*$
22	Composition function 2	$f(x) = \sum_{i=1}^n \omega_i * [\lambda_i g_i(x) + bias_i] + f^*$ $f_i^* = f_i - f_i^*, g_{1-3} = f_{14}$
23	Composition function 3	$f(x) = \sum_{i=1}^n \omega_i * [\lambda_i g_i(x) + bias_i] + f^*$ $f_i^* = f_i - f_i^*, g_{1-3} = f_{15}$
24	Composition function 4	$f(x) = \sum_{i=1}^n \omega_i * [\lambda_i g_i(x) + bias_i] + f^*$ $f_i^* = f_i - f_i^*, g_1 = f_{15}^*, g_2 = f_{12}^*, g_3 = f_9^*, \sigma = [20, 20, 20]$
25	Composition function 5	$f(x) = \sum_{i=1}^n \omega_i * [\lambda_i g_i(x) + bias_i] + f^*$ $f_i^* = f_i - f_i^*, g_1 = f_{15}^*, g_2 = f_{12}^*, g_3 = f_9^*, \sigma = [10, 30, 50]$
26	Composition function 6	$f(x) = \sum_{i=1}^n \omega_i * [\lambda_i g_i(x) + bias_i] + f^*$ $f_i^* = f_i - f_i^*, g_1 = f_{15}^*, g_2 = f_{12}^*, g_3 = f_2^*, g_4 = f_9^*, g_5 = f_{10}^*$
27	Composition function 7	$f(x) = \sum_{i=1}^n \omega_i * [\lambda_i g_i(x) + bias_i] + f^*$ $f_i^* = f_i - f_i^*, g_1 = f_{10}^*, g_2 = f_{12}^*, g_3 = f_{15}^*, g_4 = f_9^*, g_5 = f_1^*$
28	Composition function 8	$f(x) = \sum_{i=1}^n \omega_i * [\lambda_i g_i(x) + bias_i] + f^*$ $f_i^* = f_i - f_i^*, g_1 = f_{19}^*, g_2 = f_7^*, g_3 = f_{15}^*, g_4 = f_{20}^*, g_5 = f_1^*$

implemented in Matlab 2011b Unix version. The fitness error values that smaller than “eps” ($eps = 2.2204e - 016$) are considered as zeros herein.

When we test the MKE_v1 with $R * PopSize = 1$ using these benchmark functions, experiment results show that MKE_v1 has better accuracy and convergence speed for lower dimension functions and un-rotated functions than canonical PSO algorithm. In order to improve the performance, we propose an updated version named MKE_v2.

3.2. Updated version of memetic MKE Algorithm

As we know, Monkey King in Chinese famous mythological novel, Journey to the West, is the most powerful disciple of priest Sanzang. Accordingly, we appoint the global best in the population of each iteration to be Monkey King particle instead of randomly

generated ones that in MKE_v1, this version is named MKE_v2. Before the illustration of the evolution equation in MKE_v2, we give some definitions first, \hat{X} denotes the coordinate matrix of all particles in the population with the coordinate of i th particle $X_i = \{x_1, x_2, \dots, x_D\}$ to be the i th row vector of it. There are ps row vectors in \hat{X} as the population size is ps . $\hat{X}_{MK,G}$ denotes the Monkey King matrix, and there are $C \times D$ vectors in the matrix. Each vector in $\hat{X}_{MK,G}$ is $X_{MK,G}$ with the value equaling to $X_{g_{best},G}$ (the particle that has global best fitness value). Eq. (9) shows the equation of \hat{X} and $\hat{X}_{MK,G}$.

$$\hat{X} = \begin{bmatrix} X_1 \\ X_2 \\ \dots \\ X_{ps} \end{bmatrix} \hat{X}_{MK,G} = \begin{bmatrix} X_{MK,G} \\ X_{MK,G} \\ \dots \\ X_{MK,G} \end{bmatrix}_{C \times D} \tag{9}$$

Table 3
Search domain and minimum of real-parameter optimization benchmark functions.

No.	Name	Search domain	Minimum value
1	Sphere function	$[-100, 100]^D$	$f(o_1, o_2, \dots, o_d) = -1400$
2	Rotated high conditioned elliptic function	$[-100, 100]^D$	$f(o_1, o_2, \dots, o_d) = -1300$
3	Rotated bent cigar function	$[-100, 100]^D$	$f(o_1, o_2, \dots, o_d) = -1200$
4	Rotated discuss function	$[-100, 100]^D$	$f(o_1, o_2, \dots, o_d) = -1100$
5	Different powers function	$[-100, 100]^D$	$f(o_1, o_2, \dots, o_d) = -1000$
6	Rotated Rosenbrock's function	$[-100, 100]^D$	$f(o_1, o_2, \dots, o_d) = -900$
7	Rotated Schaffers F7 function	$[-100, 100]^D$	$f(o_1, o_2, \dots, o_d) = -800$
8	Rotated Ackley's function	$[-100, 100]^D$	$f(o_1, o_2, \dots, o_d) = -700$
9	Rotated Weierstrass function	$[-100, 100]^D$	$f(o_1, o_2, \dots, o_d) = -600$
10	Rotated Griewank's function	$[-100, 100]^D$	$f(o_1, o_2, \dots, o_d) = -500$
11	Rastrigin's function	$[-100, 100]^D$	$f(o_1, o_2, \dots, o_d) = -400$
12	Rotated Rastrigin's function	$[-100, 100]^D$	$f(o_1, o_2, \dots, o_d) = -300$
13	Non-continuous rotated Rastrigin's function	$[-100, 100]^D$	$f(o_1, o_2, \dots, o_d) = -200$
14	Schwefel's function	$[-100, 100]^D$	$f(o_1, o_2, \dots, o_d) = -100$
15	Rotated Schwefel's function	$[-100, 100]^D$	$f(o_1, o_2, \dots, o_d) = 100$
16	Rotated Katsuura function	$[-100, 100]^D$	$f(o_1, o_2, \dots, o_d) = 200$
17	Lunacek Bi-Rastrigin function	$[-100, 100]^D$	$f(o_1, o_2, \dots, o_d) = 300$
18	Rotated Lunacek Bi-Rastrigin function	$[-100, 100]^D$	$f(o_1, o_2, \dots, o_d) = 400$
19	Expanded Griewank's plus Rosenbrock's function	$[-100, 100]^D$	$f(o_1, o_2, \dots, o_d) = 500$
20	Expanded Scaffer's F6 function	$[-100, 100]^D$	$f(o_1, o_2, \dots, o_d) = 600$
21	Composition function1 ($n = 5$, rotated)	$[-100, 100]^D$	$f(o_1, o_2, \dots, o_d) = 700$
22	Composition function2 ($n = 3$, unrotated)	$[-100, 100]^D$	$f(o_1, o_2, \dots, o_d) = 800$
23	Composition function3 ($n = 3$, rotated)	$[-100, 100]^D$	$f(o_1, o_2, \dots, o_d) = 900$
24	Composition function4 ($n = 3$, rotated)	$[-100, 100]^D$	$f(o_1, o_2, \dots, o_d) = 1000$
25	Composition function5 ($n = 3$, rotated)	$[-100, 100]^D$	$f(o_1, o_2, \dots, o_d) = 1100$
26	Composition function6 ($n = 5$, rotated)	$[-100, 100]^D$	$f(o_1, o_2, \dots, o_d) = 1200$
27	Composition function7 ($n = 5$, rotated)	$[-100, 100]^D$	$f(o_1, o_2, \dots, o_d) = 1300$
28	Composition function8 ($n = 5$, rotated)	$[-100, 100]^D$	$f(o_1, o_2, \dots, o_d) = 1400$

Table 4
Search domain and minimum of large-scale benchmark functions.

No.	Name	Search domain	Minimum value
1	Shifted sphere function	$[-100, 100]^D$	$f(o_1, o_2, \dots, o_d) = -450$
2	Shifted Schwefel's problem 2.21	$[-100, 100]^D$	$f(o_1, o_2, \dots, o_d) = -450$
3	Shifted Rosenbrocks function	$[-100, 100]^D$	$f(o_1, o_2, \dots, o_d) = 390$
4	Shifted Rastrigins function	$[-5, 5]^D$	$f(o_1, o_2, \dots, o_d) = -330$
5	Shifted Griewanks function	$[-600, 600]^D$	$f(o_1, o_2, \dots, o_d) = -180$
6	Shifted Ackleys function	$[-32, 32]^D$	$f(o_1, o_2, \dots, o_d) = -140$
7	FastFractal "DoubleDip" function	$[-1, 1]^D$	$f(o_1, o_2, \dots, o_d)$ unknown

Now, we get back to illustrate the evolution scheme of MKE_v2. The local search/exploitation is implemented by Eq. (10). \hat{X}_{diff} is the exploitation matrix, generated by the difference of two random matrices \hat{X}_{r1} and \hat{X}_{r2} . An illustration of two dimension \hat{X}_{diff} is shown in Fig. 6. \hat{X}_{r1} and \hat{X}_{r2} are generated by randomly selecting $C \times D$ row vectors from \hat{X} (select without replacement), the earlier selected row vectors of \hat{X} appears in the front rows of \hat{X}_{r1} and \hat{X}_{r2} . FC is the fluctuation coefficient of the exploitation matrix. $\hat{X}_{MK,G+1}(i)$ denotes the i^{th} row vector of $\hat{X}_{MK,G+1}$. The next generation ($X_{MK,G+1}$) of $X_{MK,G}$ is updated by the row vector that find the optimum value among row vectors in $\hat{X}_{MK,G+1}$. The common particles still use the same update scheme mentioned in MKE_v1 with the equation shown in Eq. (7).

$$\begin{cases} \hat{X}_{diff} = (\hat{X}_{r1} - \hat{X}_{r2}) \\ \hat{X}_{MK,G+1} = \hat{X}_{MK,G} + FC * \hat{X}_{diff} \\ X_{MK,G+1} = opt\{\hat{X}_{MK,G+1}(i)\}, i = 1, 2, \dots, C \times D \end{cases} \quad (10)$$

In the conducted experiment, we use $C = 3, F = 5, FC = 0.5$ and $C = 3, F = 2, FC = 2$ to make comparisons. The optimization results of the 28 benchmark functions with 10-dimension after 10,000 generations are shown in Table 5. For each function we run 20 times and the best minimum and mean minimum of the 20-run fitness errors are listed in the table. We can see that the optimization results are not good enough only by employing the exploitation of the Monkey King particle (global best particle), though

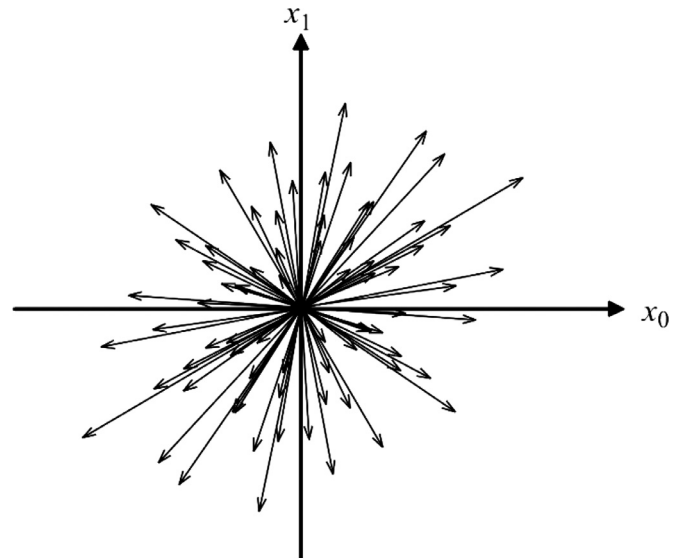


Fig. 6. The Monkey King particle's exploitation matrix/vectors.

the proposed MKE_v2 has an overall better performance than the canonical PSO algorithm. Then we propose the third Monkey King Evolution version (MKE_v3).

Table 5

Comparison results of MKE_v2 with different parameter settings. The best values of 20-run fitness errors are emphasized in **BOLDFACE** and the average values of the 20-run fitness errors are highlighted in *ITALIC* fonts.

f*-f(o)	MKE_v2, F = 2, FC = 2		MKE_v2, F = 5, FC=0.5		PSO	
	Best	Mean	Best	Mean	Best	Mean
1	2.2737E-13	3.0695E-13	0	2.2737E-14	2.0552E + 02	4.8932E + 02
2	1.2857E + 03	1.8772E + 04	5.1501E + 04	2.2937E + 05	6.8267E + 05	1.2207E + 06
3	2.8475E-01	3.1489E + 07	5.6787E + 00	6.4120E + 05	2.4609E + 08	4.9810E + 08
4	0	5.7981E-13	1.7061E + 02	1.0869E + 03	1.5163E + 03	2.3930E + 03
5	0	2.3306E-13	0	3.9792E-14	6.0427E + 01	1.0332E + 02
6	1.1461E-03	1.3162E + 01	7.6161E-02	7.9281E + 00	2.1151E + 01	3.6457E + 01
7	1.7013E + 01	7.0382E + 01	3.1573E + 00	2.5164E + 01	2.4418E + 01	3.2066E + 01
8	2.0000E + 01	2.0232E + 01	2.0179E + 01	2.0261E + 01	2.0119E + 01	2.0236E + 01
9	4.5935E + 00	6.1842E + 00	3.0581E + 00	4.5408E + 00	5.0744E + 00	6.3820E + 00
10	8.1085E-01	2.5203E + 00	6.6424E-02	6.3203E-01	2.9134E + 01	4.6083E + 01
11	4.9748E + 00	1.4775E + 01	0	3.9798E-01	3.4236E + 01	4.6911E + 01
12	1.1940E + 01	3.1291E + 01	1.5919E + 01	3.0097E + 01	2.6582E + 01	4.5151E + 01
13	2.3971E + 01	5.0830E + 01	5.9475E + 00	2.7493E + 01	3.3342E + 01	4.5142E + 01
14	1.8197E + 02	3.1795E + 02	4.0176E + 01	1.4161E + 02	7.1241E + 02	1.0740E + 03
15	3.0321E + 02	9.2910E + 02	3.6868E + 02	9.0854E + 02	1.0092E + 03	1.1380E + 03
16	2.1292E-01	4.4480E-01	2.0879E-01	5.0173E-01	5.3371E-01	7.8679E-01
17	4.7861E + 00	1.7968E + 01	1.0276E-09	8.2347E + 00	7.1864E + 01	9.6239E + 01
18	1.1805E + 01	4.5189E + 01	2.0874E + 01	4.1024E + 01	7.7151E + 01	9.0736E + 01
19	4.5331E-01	1.4621E + 00	2.6635E-01	6.8545E-01	5.7323E + 00	8.5746E + 00
20	2.8668E + 00	3.5650E + 00	1.2588E + 00	3.0732E + 00	2.7487E + 00	3.2410E + 00
21	1.0000E + 02	3.7016E + 02	2.0000E + 02	2.9009E + 02	3.7741E + 02	4.3280E + 02
22	5.5712E + 01	4.0573E + 02	1.0979E + 01	2.1706E + 02	9.4005E + 02	1.2336E + 03
23	6.2828E + 02	1.2956E + 03	2.3218E + 02	9.8438E + 02	8.1939E + 02	1.1645E + 03
24	2.0855E + 02	2.1831E + 02	2.0783E + 02	2.1347E + 02	2.1516E + 02	2.1778E + 02
25	2.0393E + 02	2.1698E + 02	2.0333E + 02	2.1361E + 02	2.1622E + 02	2.1928E + 02
26	1.1890E + 02	2.1664E + 02	1.1791E + 02	2.1767E + 02	1.3451E + 02	1.8107E + 02
27	3.4323E + 02	5.3126E + 02	4.0000E + 02	5.3236E + 02	5.0656E + 02	5.7536E + 02
28	3.0000E + 02	4.6266E + 02	1.0000E + 02	3.1299E + 02	6.0308E + 02	7.2820E + 02

3.3. Monkey King Evolution version3

In this version, common particles are equipped with the same exploitation behavior as Monkey King particles, in other words, the particles are equivalent. Exploration and exploitation are implemented simultaneously by affine-like transformation (In affine transformation, $f: X \rightarrow Y$, is of the form $X \mapsto MX + b$. Here in this paper we use a new affine-like transformation style $X \mapsto M \otimes X + Bias$, $Bias = \bar{M} \otimes B$, \otimes denotes multiplication of corresponding matrix elements, same as “.” operation in Matlab), and all particles in the population use a matrix M for this transformation. We also give some definitions before the introduction of the evolution equation in MKE_v3. $\hat{X}_{gbest,G}$ denotes the replicated global best matrix, and there are ps vectors in the matrix. Each vector in $\hat{X}_{gbest,G}$ is with the same value $X_{gbest,G}$ (the coordinate of the particle that has global best fitness value). \hat{X}_G denotes the G^{th} generation of \hat{X} , \hat{X} is the coordinate matrix, which is the same as the one mentioned in MKE_v2. Eq. (11) shows the equation of $\hat{X}_{gbest,G}$, and Eq. (12) shows the evolution scheme in MKE_v3.

$$\hat{X} = \begin{bmatrix} X_1 \\ X_2 \\ \dots \\ X_{ps} \end{bmatrix} \hat{X}_{gbest,G} = \begin{bmatrix} X_{gbest,G} \\ X_{gbest,G} \\ \dots \\ X_{gbest,G} \end{bmatrix} \quad (11)$$

$$\begin{cases} \hat{X}_{diff} = (\hat{X}_{r1} - \hat{X}_{r2}) \\ \hat{X}_{gbest,G+1} = \hat{X}_{gbest,G} + FC * \hat{X}_{diff} \\ \hat{X}_{G+1} = M \otimes \hat{X}_G + Bias \\ Bias = \bar{M} \otimes \hat{X}_{gbest,G+1} \end{cases} \quad (12)$$

\hat{X}_{diff} also denotes the exploitation matrix, but \hat{X}_{r1} and \hat{X}_{r2} have extended meanings with the ones introduced in MKE_v2. \hat{X}_{r1} and \hat{X}_{r2} here in MKE_v3 extend the row sizes from $C \times D$ to ps (the population size). In this case, the randomly selecting row vectors from \hat{X} to \hat{X}_{r1} and \hat{X}_{r2} can be implemented by randomly permutating row vectors of \hat{X} .

M is the transformation matrix, and it is transformed from a matrix M_{tmp} which is generated by the multiplication of orthogonal eigen-vector matrix P and diagonal eigen-value matrix ($M_{tmp} = P^T \Lambda P$). $\Lambda = diag(d_1, d_2, \dots, d_D)$ is a diagonal eigen-value matrix and used for the amplification of the difference matrix. For simplicity, M_{tmp} is initialized by a lower triangular matrix with the elements equaling to ones. Eq. (13) gives an example to show the transformation for M_{tmp} to M with particle population size equaling to D . There are two steps for the transformation, the first step is to randomly permute the elements of each D -dimension row vector in M_{tmp} , and the second step is to randomly permute the row vectors with the elements of each row vector unchanged, so we can get M .

$$M_{tmp} = \begin{bmatrix} 1 & & & \\ 1 & 1 & & \\ & & \dots & \\ 1 & 1 & \dots & 1 \end{bmatrix} \sim \begin{bmatrix} 1 & 1 & & \\ & \dots & & \\ 1 & 1 & \dots & 1 \\ & & & 1 \end{bmatrix} = M \quad (13)$$

Usually, the size of the particle population is larger than particle coordinate dimension, matrix M_{tmp} needs to be extended according to population size ps . For example, when $ps = 2D$, M_{tmp} is extended to duplicated matrix shown in Eq. (14). Generally, when $ps \% D = k$, the first k rows of the $D \times D$ lower triangular matrix are

included in M_{tmp} , and M is adaptive with the change of M_{tmp} .

$$M_{tmp} = \begin{bmatrix} 1 & & & & & & \\ 1 & 1 & & & & & \\ & & \dots & & & & \\ 1 & 1 & & \dots & 1 & & \\ 1 & 1 & & & & & \\ & & & \dots & & & \\ 1 & 1 & & \dots & 1 & & \\ & & & & & & \\ 1 & 1 & & \dots & 1 & & \end{bmatrix} \sim \begin{bmatrix} 1 & 1 & & & & & \\ & \dots & & & & & \\ 1 & 1 & & \dots & 1 & & \\ & & \dots & & & & \\ 1 & \dots & & 1 & & & \\ & & & 1 & & & \\ 1 & & & \dots & 1 & & \\ & & & & & & \\ 1 & & & \dots & 1 & & \end{bmatrix} = M \quad (14)$$

\bar{M} is binary reverse operation of M . The corresponding values of non-zero elements in M are zeros in matrix \bar{M} while the corresponding values of zero elements are ones. Eq. (15) shows an example of binary reverse operation.

$$M = \begin{bmatrix} 1 & & & & & & \\ 1 & 1 & & & & & \\ & & \dots & & & & \\ 1 & 1 & & \dots & 1 & & \\ & & & \dots & & & \\ 1 & 1 & & \dots & 1 & & \end{bmatrix}, \bar{M} = \begin{bmatrix} 0 & 1 & \dots & 1 \\ 0 & 0 & \dots & 1 \\ & & \dots & 1 \\ 0 & 0 & \dots & 0 \end{bmatrix} \quad (15)$$

The pseudo-code of the algorithm is shown in Algorithm 2. So

Algorithm 2 Pseudo-code of the final version of Monkey King Evolution Algorithm.

Initialization:
Initialize the searching space $R(r_1, r_2, \dots, r_d)$ and the benchmark function $f(X)$.

Iteration:

- 1: **while** $exeTime < MaxIteration$ **do**
- 2: **if** $exeTime = 1$ **then**
- 3: Generate the population coordinates $X_i = (x_{i,1}, x_{i,2}, \dots, x_{i,d})^T$ and generate the Monkey King particle and change its flag.
- 4: **end if**
- 5: **if** $exeTime > 1$ **then**
- 6: Particles evolution in Eq. (12).
- 7: **end if**
- 8: Calculate the fitness value and update X_{pbest} .
- 9: Record the optima coordinate X_{gbest} .
- 10: **end while**

Output:
The global optima X_{gbest} and $f(X_{gbest})$.

there is only one parameter FC of the proposed MKE_v3 should be determined in experiment. Empirically, $FC = 0.7$ is a good choice for MKE_v3, and we use this setting in the following experiment.

4. Model of routing and fuel consumption in grid networks

Vehicle routing and scheduling models are very useful for the dynamic vehicle transportation in urban area. Real time information of street length, lanes, vehicle density, direction, velocity restrictions of a certain road are all recorded and collected by WSNs with the aid of city traffic surveillance system and global positioning system for vehicle identification and navigation. The actual need of routing is to search the optimal way from a source to a destination that satisfy the driver's needs. We mainly analyze the shortest path (Dijkstra method), A* algorithm and our proposed algorithm for finding a least gasoline consumption path of a navigation. A simulation is conducted on a grid network, and the navigation result is shown in Fig. 7.

The routing of vehicles can be modeled as a directed graph $G = (V, E)$ which consists of road (Edge) and intersections (Node). In Fig. 7, we randomly generated 10,000 vehicles to simulate the traffic condition that should usually be collected by WSNs at some time in a day, the road is 2-lane road, and the distance between

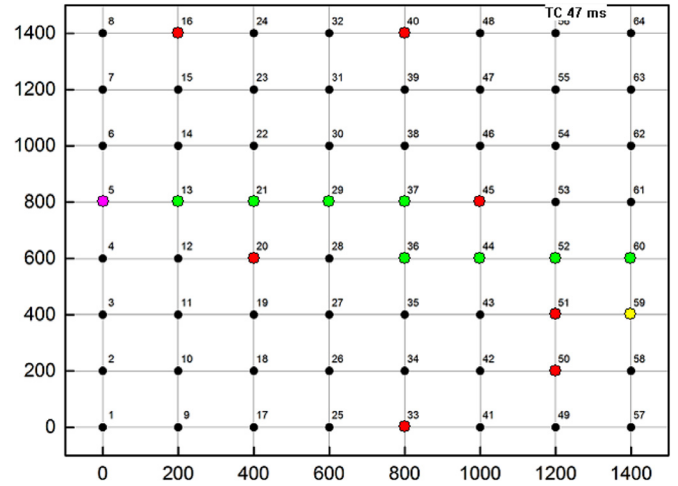


Fig. 7. Grid network of vehicle navigation simulation. (For interpretation of the references to color in this figure, the reader is referred to the web version of this article).

two nodes in the grid network is 2 km, and there are 51 vehicles, on average, in one square kilometer with 89 vehicles on one section of road (2 km, the interval between two intersections). When the density is 1.2 times than normal traffic density or more than 3 vehicles in a 22.4 m, we denote the section congested with a delay time T_0 . The red points in Fig. 7 is congestion nodes, the pink one is the generated start point, the yellow one is the destination and green ones are the intersections nodes which demonstrate the optimal navigation result. The gasoline consumption fitness function is a function of edge-travel gasoline cost and congestion gasoline cost. The fitness function is shown in Eq. (16). $g(\omega_i)$ denotes the congestion weight of $Edge_i$ and $E_{i, cost}$ denotes the gasoline cost on $Edge_i$ with the delay gasoline cost on the following intersection included.

$$f = \sum_{i=1}^n g(\omega_i) * E_{i, cost} \quad (16)$$

Each navigation path candidate is composed of node-edge sequence, and we use the node sequence herein the paper to denote each candidate path for simplicity. For example, X_i in gasoline consumption optimization is the i -th node sequence that contains the i -th candidate path of the navigation. All the intersection nodes of the local area are labeled for the navigation and different permutation sequence of these nodes that contain a candidate path of a navigation is a potential solution. For the definition of E_i , i is the start node of the edge, and edge is named after the start node. Therefore, $E_{i, cost}$ is the gasoline cost of E_i . $g(\omega_i)$ is the congestion weight of edge E_i , and the value is determined by the traffic condition at some time in a day. For simplicity, the value of $g(\omega_i)$ is set to 0 when the i -th intersection node is not on a navigation path, and the value is set to 1 when it is on a navigation path. $E_{i, cost}$ in the navigation path can be calculated with the congestion weight $g(\omega_i)$ constructed according to the current traffic condition.

Recent research shows that there is an optimum velocity range for each car. In our experiment, a typical small gasoline ($< 1400 \text{ cm}^3$) Euro 4 passenger car is used for the analysis. Fig. 8² shows the optimal speed for minimum fuel consumption. From the chart, we can separate four areas with corresponding four velocity ranges. The first range is 0–30 kph with quite high fuel consumption. This speed range is typical for cars traveling in a city with

² <http://www.myengineeringworld.net/2012/05/optimal-speed-for-minimum-fuel.html>.

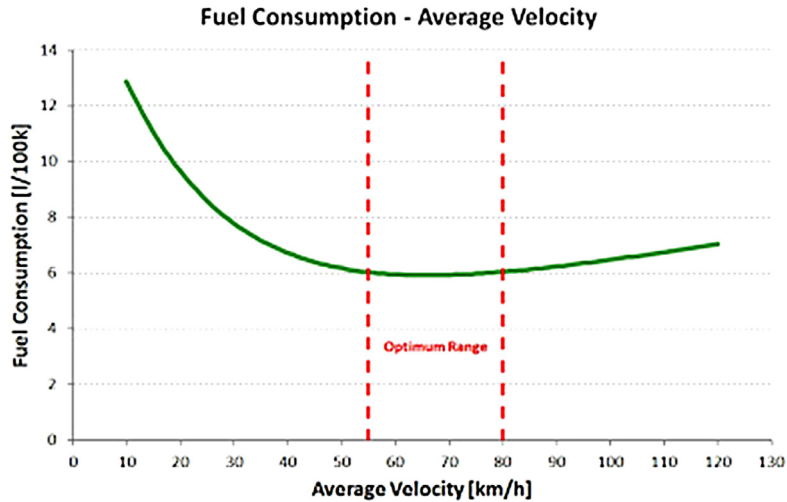


Fig. 8. Fuel consumption under different velocities.

Table 6
Parameters setting of different algorithms.

Algorithms.	Parameters settings
iwPSO	$c1 = c2 = 2.0, iw = 0.5, vel = rnd$
ccPSO	$c1 = c2 = 2.05, iw = 1, K = 0.7298, v = rnd$
ccPSO-local	$c1 = c2 = 2.05, iw = 1, K = 0.7298, VonNeumannTopology, v = rnd$
FIPS_Uring	$cc = 4.1, iw = 1, K = 0.7298, UringTopology, v = rnd$
CLPSO	$iw = 0.9 \cdot 0.3, cc = 1, 49455, Pc = 0 \cdot 0.5, stay_num = 7, v = rnd, vmax = 0.2R$
DNLPSO	$c1 = c2 = 1.49445, iw = 0.9 \cdot 0.4, Pc = 0.45 \cdot 0.05, m = 3, g = 5, v = rnd$
MKE_v3	$F = 0.7$

Table 7
Comparison results of best minimum error in 20 runs with the same number of function evaluations. The best result of each function is emphasized in boldface and the best draw results of each function is highlighted in italic fonts.

D = 10.	iwPSO	ccPSO	ccPSO-local	FIPS_Uring	CLPSO	DNLPSO	MKE_v3
1	0	0	0	0	0	0	0
2	6.2219E + 03	1.5341E + 03	3.1794E + 03	2.5958E + 05	1.0742E + 05	1.1018E + 05	0
3	8.4504E + 00	7.2012E-03	6.3060E-04	9.3183E + 02	2.5073E + 04	3.1346E + 00	0
4	2.3032E-03	1.0383E-05	2.9173E + 01	1.0442E + 03	8.1690E + 02	0	0
5	0	0	0	0	0	0	0
6	1.0407E-01	5.8119E-03	1.3798E-03	2.9921E + 00	5.7861E-02	0	0
7	5.4720E-01	7.6699E-01	1.1821E-01	1.4475E-01	2.7911E + 00	7.1501E-01	2.6603E-11
8	2.0109E + 01	2.0142E + 01	2.0150E + 01	2.0135E + 01	2.0108E + 01	2.0000E + 01	2.0227E + 01
9	7.5826E-01	6.6214E-01	1.3548E + 00	1.1535E + 00	2.0014E + 00	2.0241E + 00	0
10	1.4268E-01	9.1142E-02	4.9242E-02	5.4348E-01	1.4956E-01	3.6914E-02	4.917E-02
11	0	9.9496E-01	0	1.2921E + 00	0	7.9597E + 00	0
12	5.9698E + 00	4.9748E + 01	2.9849E + 00	1.9855E + 01	2.2242E + 00	4.97480E + 00	4.7202E + 00
13	6.3297E + 00	3.1902E + 00	3.0705E + 00	2.1921E + 01	4.1636E + 00	2.0994E + 00	9.9496E-01
14	3.5399E + 00	3.5399E + 00	3.4774E + 00	2.4878E + 02	0	1.3679E + 00	3.8531E + 00
15	1.6691E + 02	8.6620E + 01	5.0109E + 01	8.4792E + 02	3.4004E + 02	3.2461E + 02	2.9998E + 02
16	1.4823E-01	1.5265E-01	2.0954E-01	5.9590E-01	3.0200E-01	1.1897E-01	6.1354E-01
17	1.0312E + 01	1.7721E + 00	1.0648E + 01	2.4666E + 01	2.9551E + 00	1.1719E + 01	3.9636E-01
18	1.2079E + 01	4.8434E + 00	9.1023E + 00	3.0724E + 01	1.7227E + 01	1.9027E + 01	1.2956E + 01
19	3.0890E-01	6.9478E-02	1.5285E-01	1.3817E + 00	5.3203E-02	5.9673E-01	2.8013E-01
20	8.9747E-01	1.7711E + 00	1.8324E + 00	2.5448E + 00	1.9101E + 00	2.2711E + 00	1.0848E + 00
21	2.0000E + 02	2.0000E + 02	2.0000E + 02	2.3411E + 02	4.1487E + 00	2.0099E + 02	1.0000E + 02
22	1.7541E + 01	4.1743E + 01	4.3246E + 02	1.9707E + 02	5.1004E + 00	4.0347E + 02	3.1368E + 01
23	3.8489E + 02	3.7587E + 02	1.1655E + 02	1.0744E + 03	2.6518E + 02	6.1120E + 02	1.5338E + 02
24	2.0083E + 02	1.2547E + 02	2.0402E + 02	2.0680E + 02	1.1540E + 02	2.0080E + 02	1.1186E + 02
25	2.0154E + 02	2.0071E + 02	1.1359E + 02	2.0814E + 02	1.2615E + 02	2.0197E + 02	2.0000E + 02
26	1.0398E + 02	1.0497E + 02	1.0796E + 02	1.2904E + 02	1.0890E + 02	1.0717E + 02	1.0398E + 02
27	3.0110E + 02	3.5496E + 02	3.1318E + 02	3.5674E + 02	2.9236E + 02	3.0574E + 02	3.0000E + 02
28	3.0000E + 02	3.0000E + 02	1.0000E + 02	1.5215E + 02	1.1715E + 02	3.0000E + 02	1.0000E + 02
Win	2	1	1	0	5	2	8
Draw	4	2	4	2	3	4	7
Total	6	3	5	2	8	6	15

Table 8

Comparison results of median value of minimum errors in 20 runs with the same number of function evaluations. The best result of each function is emphasized in boldface and the best draw results of each function is highlighted in italic fonts.

$D = 10$.	iwPSO	ccPSO	ccPSO-local	FIPS_Uring	CLPSO	DNLPSO	MKE_v3
1	0	0	0	0	0	0	0
2	6.3806E + 04	1.2153E + 04	1.2396E + 04	8.4876E + 05	2.6012E + 05	1.2038E + 06	0
3	1.1851E + 05	1.9421E + 04	7.1724E + 00	4.8359E + 03	2.0317E-05	1.4948E + 06	0
4	3.1851E-02	4.8394E-05	1.3013E + 01	2.1160E + 03	1.8513E + 03	1.5871E-01	0
5	1.1369E-13	1.1369E-13	0	0	0	0	0
6	9.8661E-00	7.7159E-02	3.5043E-02	3.9566E + 00	1.7674E-01	1.0751E + 01	0
7	1.1007E + 01	1.0536E + 01	5.3617E-01	5.7795E-01	5.6673E + 00	1.9268E + 01	9.5508E-03
8	2.0232E + 01	2.0228E + 01	2.0215E + 01	2.0249E + 01	2.0228E + 01	2.0278E + 01	2.0478E + 01
9	4.3370E + 00	3.2650E + 00	2.8860E + 00	3.8699E + 00	2.9952E + 00	4.7871E + 00	1.7891E + 00
10	4.5315E-01	3.3458E-01	1.7096E-01	6.2779E-01	3.1643E-01	3.0889E-01	1.6972E-01
11	1.9899E + 00	2.9849E + 00	0	9.3415E + 00	0	1.1442E + 01	2.9849E + 00
12	1.4924E + 01	1.4924E + 01	7.9597E + 00	2.8003E + 01	6.0776E + 00	1.5919E + 01	1.3718E + 01
13	1.7901E + 01	1.9618E + 01	9.5999E + 00	2.7402E + 01	9.4278E + 00	3.2975E + 01	1.9161E + 01
14	1.3632E + 02	1.9300E + 02	6.1817E + 01	5.6135E + 02	0	4.0847E + 02	1.0360E + 02
15	6.6693E + 02	7.8020E + 02	4.1893E + 02	1.1650E + 03	5.1257E + 02	6.7508E + 02	1.0711E + 03
16	4.9882E-01	3.5291E-01	5.8408E-01	7.8826E-01	6.7720E-01	4.2404E-01	1.1980E + 00
17	1.1121E + 01	1.1216E + 01	1.2122E + 01	3.3599E + 01	6.9076E + 00	2.4595E + 01	1.1374E + 01
18	2.2190E + 01	1.9620E + 01	1.4389E + 01	3.7075E + 01	2.1799E + 01	3.5661E + 01	3.2489E + 01
19	5.4808E-01	5.3011E-01	4.8039E-01	2.0450E + 00	1.7803E-01	9.6366E-01	6.2560E-01
20	3.0051E + 00	3.4763E + 00	2.2794E + 00	2.7727E + 00	2.5118E + 00	2.8935E + 00	1.7738E + 00
21	4.0019E + 02	4.0019E + 02	4.0019E + 02	4.0019E + 02	1.7552E + 02	4.0019E + 02	4.0019E + 02
22	2.1424E + 02	2.0658E + 02	1.5606E + 02	5.4100E + 02	1.0606E + 02	7.5007E + 02	1.4334E + 02
23	9.5811E + 02	9.1353E + 02	4.1825E + 02	1.2283E + 03	5.7812E + 02	1.0252E + 03	9.9672E + 02
24	2.1004E + 02	2.0893E + 02	2.0708E + 02	2.1175E + 02	1.2189E + 02	2.1218E + 02	2.0700E + 02
25	2.1437E + 02	2.0808E + 02	2.0722E + 02	2.1309E + 02	1.4166E + 02	2.1669E + 02	2.0654E + 02
26	2.0002E + 02	2.0002E + 02	1.2391E + 02	1.3871E + 02	1.2402E + 02	2.0004E + 02	1.2292E + 02
27	5.0404E + 02	4.0503E + 02	3.7359E + 02	4.1487E + 02	3.2453E + 02	3.6286E + 02	3.0000E + 02
28	3.0000E + 02	3.0000E + 02	3.0000E + 02	3.0000E + 02	1.4265E + 02	3.0000E + 02	3.0000E + 02
Win	0	2	4	0	10	0	10
Draw	1	1	3	2	3	2	2
Total	1	3	7	2	13	2	12

continuous start and stop motion, and the traffic situation in this range is often considered as being in traffic congestion. The second range is 30–55 kph, and this velocity range is very common of a car in sub-urban or rural areas. The third range is 55–80 kph, and this is the optimum velocity range that minimize the fuel consumption. The last range is 80–120 kph, and the fuel consumption augments with the velocity increase. $E_{i, cost}$ is calculated by the addition of traveling cost and delay cost with the equation shown in Eq. (17). $Cost(vel)$ denotes the gasoline cost with corresponding velocity value vel , $Dist(vel)$ denotes the traveling distance with the velocity equaling to vel , $Cost(delay)$ denotes the engine idling fuel consumption, and T_0 is the delay time mentioned earlier in the paper.

$$\begin{cases} E_{i, cost} = E_{i, travel} + E_{i, delay} \\ E_{i, travel} = Dist(vel) * Cost(vel) \\ E_{i, delay} = T_0 * Cost(delay) \end{cases} \quad (17)$$

5. Performance evaluation and comparisons

The proposed algorithm has been tested under benchmark functions listed in Section 3.1, and contrasted with the canonical PSO, Inertia Weighted PSO (iwPSO), Constriction Coefficients PSO (ccPSO), Fully Informed Particle Swarm (FIPS), Comprehensive Learning PSO (CLPSO), Dynamic Neighborhood Learning PSO (DNLPSO). The compared algorithms herein has similar time and space complexity, so it's easier to examine the performance of these state-of-the-art algorithms. The parameter setting of PSO is $C1 = C2 = 2$, parameters of iwPSO in the experiment are $C1 = C2 = 2$ and $iw = 0.5$. For ccPSO, we use $C1 = C2 = 2.05$ and $K = 0.729$, ccPSO-local denotes constriction coefficients PSO with von Neumann topology/neighborhood, and the parameter settings are the same as ccPSO. For CLPSO, the parameters settings are $cc = 1.49445$ and $iw \in [0.2, 0.9]$ a decreasing function of iterations

and for DNLPSO, the parameters $c1 = 1.49445$, $c2 = 1.49445$, $iw \in [0.4, 0.9]$, also a decreasing function of iterations, $m = 3$ and $g = 5$.³ All the settings of algorithms are listed in Table 6, and they are the authors' recommended settings.

In our implementation, we use 100 particles of a population, and run 20 times, with 10,000 iterations in each run to get the minimum of the benchmark functions from CEC2013 and BBoB2009. The benchmark functions are used as black box test functions. The best, median, mean/standard deviation of fitness error $f - f(o)$ comparison of a single 20-run by different algorithms are shown in Tables 7–9 accordingly. As can be seen from Table 7, the fitness error of the benchmark values (f1, f4, f5, f6, f11, f26 and f28) are equal with different state-of-the-art algorithms. Illustration of the convergence speed are analyzed and shown in the following figures (Figs. 9–15) when different state-of-the-art algorithms have the same best fitness error values. For the PSO variants, there are two different ways of velocity initialization, one is that velocities initialized with zeros and the other is initialized with random values, we use random velocity values in this paper. All the compared algorithms can find the minimum value on function 1 and function 5 of CEC2013 benchmark functions. DNLPSO can find the minimum values of function 4 and function 6. Only CLPSO can find the optimum of function 14. Our algorithm has an overall better performance, and we can see that our method not only has better optimization result but also has better convergence speed.

For the validation of large scale property of the proposed algorithm, we use the benchmarks CEC2008 for the test. The dimension of the test-bed is 100- D with a population of 500 particles, and we run 20 times with 10,000 iterations each. The parameters

³ CLPSO code and DNLPSO code are from Prof. Ponnuthurai Nagaratnam Suganthan.

Table 9
Comparison results of mean/standard deviation of 20 runs with the same population size and number of function evaluations. The best result of each function is emphasized in boldface and the best draw results of each function is highlighted in italic fonts.

D = 10.	iwPSO	ccPSO	ccPSO-local	FIPS_Uring	CLPSO	DNLPSO	MKE_v3
1	<i>0/0</i>	5.6843E-14/1.0101E-13	<i>0/0</i>	<i>0/0</i>	<i>0/0</i>	<i>0/0</i>	<i>0/0</i>
2	1.4943E + 05/1.8848E + 05	2.0926E + 04/1.8434E + 04	1.1760E + 04/6.7065E + 03	8.3832E + 04/2.6904E + 05	2.8651E + 05/1.8413E + 05	1.6864E + 06/1.4808E + 06	2.2291E-14/6.8285E-14
3	1.6815E + 06/3.7625E + 07	4.1743E + 06/1.3409E + 07	1.7124E + 04/3.1723E + 04	7.7936E + 03/8.5641E + 03	2.4655E + 05/1.9573E + 05	4.5197E + 07/6.3876E + 07	4.5959E-03/1.7094E-02
4	4.5055E-02/5.1575E-02	1.9519E-04/5.7786E-04	1.8738E + 01/1.7684E + 01	1.9171E + 03/5.1283E + 02	1.8794E + 03/5.9328E + 02	1.7469E + 03/3.8752E + 03	<i>0/0</i>
5	7.9580E-14/5.3451E-14	7.9580E-14/5.3451E-14	<i>0/0</i>	<i>0/0</i>	<i>0/0</i>	1.4947E-04/3.6081E-04	<i>0/0</i>
6	9.3974E + 00/1.1850E + 01	8.6092E-01/1.6391E + 00	3.3451E-02/8.3316E-03	3.8760E + 00/5.3077E-01	1.6891E-01/7.0527E + 00	8.4198E + 00/4.6314E + 00	4.4064E + 00/4.8315E + 00
7	1.7024E + 01/2.2056E + 01	1.4176E + 01/1.1512E + 01	8.1367E-01/6.5693E-01	6.2262E-01/3.8467E-01	5.9567E + 00/1.7804E-00	2.3676E + 01/2.4266E + 01	7.0377E-01/2.1134E + 00
8	2.0213E + 01/5.7059E-02	2.0215E + 01/4.4255E-02	2.0225E + 01/4.5937E-02	2.0248E + 01/4.3374E-02	2.0220E + 01/5.7137E-02	2.0293E + 01/1.3372E-01	2.0457E + 01/8.6529E-02
9	3.9963E + 00/1.5776E + 00	3.4175E + 00/1.7503E + 00	2.8786E + 00/7.6713E-01	3.6769E + 00/9.6284E-01	3.0316E + 00/5.0727E-01	4.8402E + 00/2.3086E + 00	2.0279E + 00/1.4519E + 00
10	4.9989E-01/2.9794E-01	3.4498E-01/1.8439E-01	1.0603E-01/4.2707E-02	6.3301E-01/5.8004E-02	3.1259E-01/8.0042E-02	3.1971E-01/1.7876E-01	1.0887E-01/8.8611E-02
11	1.7411E + 00/9.6167E-01	3.5321E + 00/3.1210E + 00	9.4521E-01/1.3103E + 00	8.6795E + 00/3.8262E + 00	<i>0/0</i>	1.3929E + 01/6.0974E + 00	3.1214E + 00/1.8022E + 00
12	1.5064E + 01/5.5123E + 00	1.6018E + 01/6.7319E + 00	7.4621E + 00/2.7675E + 00	2.7347E + 01/3.4252E + 00	5.5914E + 00/1.6202E + 00	1.6940E + 01/8.9530E + 00	1.3973E + 00/6.7581E + 00
13	1.8486E + 01/6.5294E + 00	1.8805E + 01/8.2768E + 00	9.5653E + 00/3.7863E + 00	2.6346E + 01/2.6632E + 00	9.1990E + 00/2.2030E + 00	3.3325E + 01/9.9912E + 00	1.9075E + 01/8.6885E + 00
14	1.2585E + 02/9.7576E + 01	1.9985E + 02/1.3954E + 02	7.5352E + 01/5.9251E + 01	5.2574E + 02/1.1005E + 02	<i>0/0</i>	4.6460E + 00/3.1100E + 02	1.2560E + 02/1.1706E + 02
15	6.3640E + 02/2.5248E + 02	7.0312E + 02/2.7108E + 02	4.1751E + 02/1.8087E + 02	1.1446E + 03/1.1770E + 02	5.0029E + 02/8.4036E + 01	7.2541E + 02/2.4893E + 02	1.0043E + 03/3.2006E + 02
16	4.6522E-01/1.9935E-01	3.6472E-01/1.3443E-01	5.3634E-01/2.6191E-01	7.9505E-01/1.1954E-01	6.7487E-01/1.1728E-01	5.2585E-01/3.6842E-01	1.2167E + 00/3.5948E-01
17	1.1447E + 01/1.0458E + 00	1.1175E + 01/2.5557E + 00	1.1895E + 01/8.1751E-01	3.2791E + 01/4.0305E + 00	8.7542E + 00/2.4160E + 00	2.4732E + 01/9.2094E + 00	8.7361E + 00/4.3007E + 00
18	2.2961E + 01/6.7136E + 00	1.9318E + 01/5.4133E + 00	1.4284E + 01/1.7113E + 00	3.7400E + 01/3.6474E + 00	2.1875E + 01/2.2914E + 00	3.3354E + 01/8.1537E + 00	3.1576E + 01/8.1329E + 00
19	5.6809E-01/1.7289E-01	5.3041E-01/1.8566E-01	4.7916E-01/1.3250E-01	1.9199E + 00/3.0975E-01	1.5905E-01/6.0262E-02	1.0548E + 00/4.1233E-01	5.9791E-01/1.3885E-01
20	2.9026E + 00/6.2217E-01	3.1800E + 00/6.9959E-01	2.2852E + 00/2.6666E-01	2.7682E + 00/1.2796E-01	2.4449E + 00/2.5033E-01	2.9047E + 00/4.9488E-01	1.8075E + 00/5.01660E-01
21	3.6515E + 02/7.4593E + 01	3.9018E + 02/4.4765E + 01	3.6015E + 02/8.2158E + 01	3.7925E + 02/5.0392E + 01	1.5772E + 02/5.4660E + 01	3.8027E + 02/6.2994E + 01	3.7665E + 02/7.37814E + 01
22	1.9603E + 02/1.1511E + 02	2.4275E + 02/1.4075E + 02	1.2397E + 02/6.9959E + 01	5.0891E + 02/1.4705E + 02	1.0170E + 01/3.1552E + 00	8.0999E + 02/3.1771E + 02	9.0642E + 01/7.4036E + 01
23	8.6764E + 02/2.6074E + 02	8.7279E + 02/2.6247E + 02	3.9136E + 02/1.8061E + 02	1.2210E + 03/9.4446E + 01	5.6413E + 02/1.0927E + 02	1.2058E + 03/4.1367E + 02	9.8230E + 02/3.1535E + 02
24	2.1030E + 02/4.8793E + 00	2.0488E + 02/1.9098E + 02	2.0705E + 02/2.3643E + 01	2.1135E + 02/2.2063E + 00	1.2301E + 02/6.2363E + 00	2.1017E + 02/5.8131E + 00	2.0435E + 02/1.3891E + 01
25	2.1379E + 02/4.4460E + 00	2.0884E + 02/3.8450E + 00	2.0262E + 02/2.1029E + 01	2.1301E + 02/2.1873E + 00	1.4422E + 02/1.5950E + 01	2.1381E + 02/7.4104E + 00	2.0553E + 02/5.2419E + 00
26	1.8572E + 02/6.8971E + 01	1.8191E + 02/7.0252E + 01	1.1399E + 02/4.0556E + 00	1.4337E + 02/1.5172E + 01	1.1447E + 02/3.4262E + 00	1.6795E + 02/4.2324E + 01	1.1269E + 02/4.9148E + 00
27	4.5301E + 02/1.0575E + 02	4.0298E + 02/2.8914E + 01	3.6789E + 02/2.6391E + 01	4.1130E + 02/2.7367E + 01	3.2683E + 02/1.1156E + 01	4.1441E + 02/1.2431E + 02	3.2536E + 02/8.9250E + 01
28	3.0000E + 02/0	3.5024E + 02/1.2687E + 02	2.6000E + 02/8.2078E + 01	2.9260E + 02/3.3060E + 01	1.4081E + 02/1.7011E + 01	3.0000E + 02/0	2.9608E + 02/2.8006E + 01
Win	1	0	3	0	10	0	10
Draw	1	2	2	2	2	1	2
Total	2	2	5	2	12	1	12

Table 10

Comparison results of best error in 20-Run under CEC2008 large scale benchmark functions. The best result of each function is emphasized in boldface and the best draw results of each function is highlighted in italic fonts.

$D = 100.$	iwPSO	ccPSO	ccPSO-local	FIPS_Uring	CLPSO	DNLPSO	MKE_v3
1	3.2999E + 02	3.4106E-13	5.6843E-14	5.5998E + 04	1.8568E-01	2.3194E + 04	5.6843E-14
2	1.7190E + 01	9.8553E + 00	3.0533E + 01	7.4635E + 01	5.1614E + 01	1.3400E + 01	1.0605E + 00
3	8.9575E + 01	5.8249E + 01	5.1414E + 01	1.1137E + 10	1.0264E + 04	3.1491E + 07	7.6646E + 00
4	5.9698E + 01	2.0795E + 02	1.8805E + 02	9.5246E + 02	2.4207E-01	3.4027E + 02	1.1243E + 02
5	1.1728E + 00	1.9895E-13	2.8422E-14	5.4630E + 02	1.3149E-02	1.7210E + 02	2.8422E-14
6	1.7053E-13	1.2139E + 00	1.1369E-13	1.7362E + 01	1.3610E + 00	1.1482E + 01	5.6843E-14
7	-1.4703E + 03	-1.4345E + 03	-1.4028E + 03	-9.2989E + 02	-1.4314E + 03	-1.3972E + 03	-1.4733E + 03
Win	0	0	0	0	1	0	4
Draw	0	0	2	0	0	0	2
Total	0	0	2	0	1	0	6

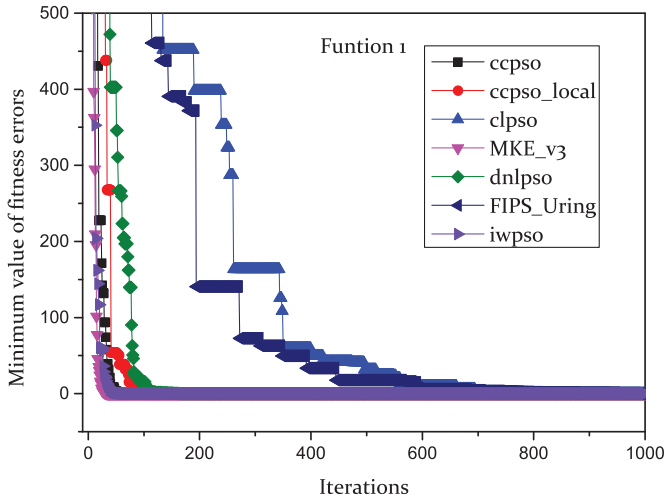


Fig. 9. Comparison of different algorithm on function 1.

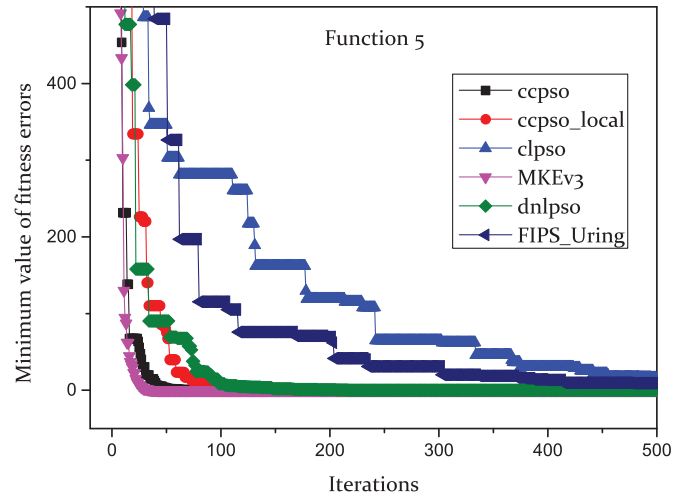


Fig. 11. Comparison of different algorithms on function 5.

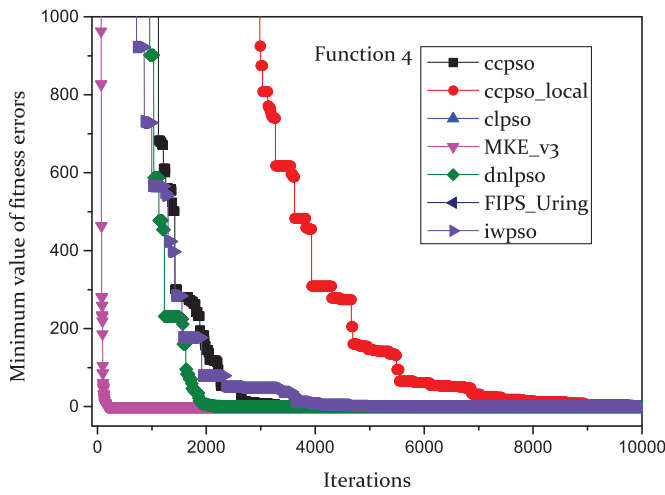


Fig. 10. Comparison of different algorithms on function 4.

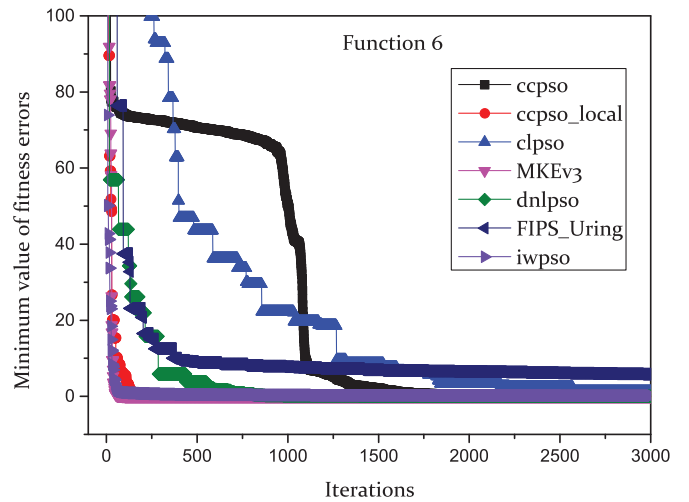


Fig. 12. Comparison of different algorithms on function 6.

setting is the best fitted ones of each algorithm with the same values mentioned above. The benchmarks are also used as black-box test functions and the best minimum error $f - f(o)$ comparisons are shown in Tables 10 and 11. Experiment results show that our algorithm outperforms others significantly. When we use this algorithm to find the least fuel consumption path of a traffic navigation application. It also performs very well. The least fuel consumption comparison with Dijkstra and A* are shown in Tables 12 and 13, respectively. They show the average fuel consumption and time consumption of 1000 times navigation of A* algorithm and

shortest path algorithm (Dijkstra) separately by comparison with our algorithm respectively. We can see that the proposed algorithm outperforms on gasoline consumption over the two algorithms.

6. Conclusion

In this paper, we propose Monkey King Evolutionary algorithm, analyze three update schemes, and then we use benchmark functions to validate the proposed algorithm. Comparisons are made between our algorithm and state-of-the-art PSO variants,

Table 11
Comparison results of mean/standard deviation of 20-Run under CEC2008 large scale benchmark functions. The best result of each function is emphasized in boldface and the best draw results of each function is highlighted in italic fonts.

D = 100.	iwPSO	ccPSO	ccPSO-local	FIPS_Uring	CLPSO	DNLPSO	MKE_v3
1	5.0213E + 02/1.9858E + 02	6.2272E-12/2.3814E-11	5.6843E-14/0	6.4343E + 04/4.0084E + 03	2.4108E-01/3.4933E-02	3.7632E + 04/7.6068E + 03	5.6843E-14/0
2	1.9779E + 01/1.5107E + 00	1.5239E + 01/3.4556E + 00	3.4727E + 01/2.1694E + 00	7.8362E + 01/1.8366E + 00	5.5218E + 01/1.6941E + 00	3.4593E + 01/2.2565E + 01	2.3887E + 00/8.6376E-01
3	1.9831E + 02/7.5928E + 01	1.2882E + 02/4.7968E + 01	1.0531E + 02/2.2950E + 01	1.5521E + 10/1.8620E + 09	1.2970E + 04/1.4926E + 03	3.3977E + 08/3.0349E + 08	4.2294E + 01/2.8079E + 00
4	9.4666E + 01/2.1137E + 01	3.6256E + 02/6.1874E + 01	2.3760E + 02/2.1194E + 01	1.0388E + 03/2.8191E + 01	1.0011E + 00/3.0238E-01	6.9793E + 02/2.2689E + 02	1.3934E + 02/2.1267E + 01
5	2.3165E + 01/6.8379E-01	2.8839E-02/4.2480E-02	2.8422E-14/0	5.9223E + 02/2.4041E + 01	2.0526E-02/3.6250E-03	2.7306E + 02/8.2656E + 01	4.9286E-04/2.2042E-03
6	2.1316E-13/2.3510E-14	3.8780E + 00/5.4874E + 00	1.3216E-13/1.3908E-14	1.7855E + 01/1.8430E-01	1.5787E + 00/1.2246E-01	1.4561E + 01/1.8313E + 00	8.5265E-14/2.3099E-14
7	-1.4265E + 03/1.9730E + 01	-1.4005E + 03/2.1153E + 01	-1.3820E + 03/1.2110E + 01	-8.9138E + 02/1.6633E + 01	-1.4219E + 03/6.5729E + 01	-1.3972E + 03/1.3744E + 02	-1.4398E + 03/2.0909E + 01
Win	0	0	1	0	0	0	4
Draw	0	0	1	0	1	0	1
Total	0	0	2	0	1	0	5

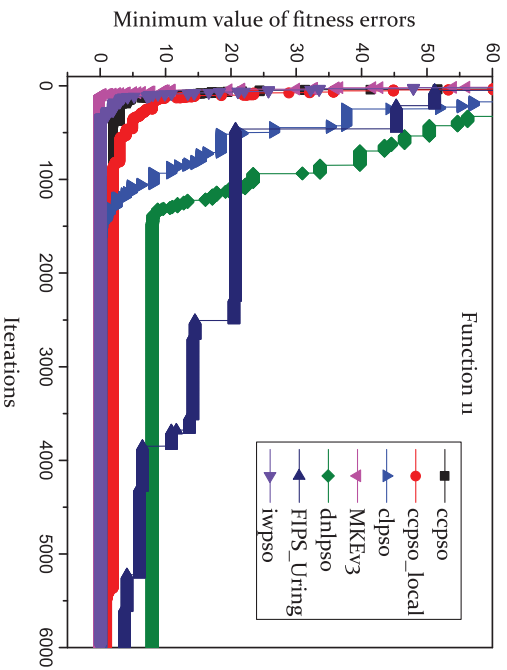


Fig. 13. Comparison of different algorithms on function 11.

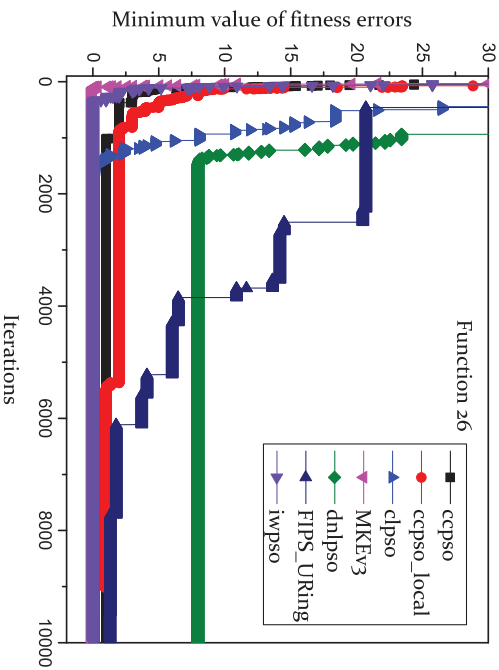


Fig. 14. Comparison of different algorithms on function 26.

Table 12
The comparison of average fuel consumption and time consumption for 1000 times navigation between Dijkstra and our algorithm.

Algorithms	Ave time cost (h)	Ave fuel cost
Dijkstra	1.2	$1.1 \times 10^{-2} L$
MKE_v3	0.9	$9.8 \times 10^{-3} L$

Table 13
The comparison of average fuel consumption and time consumption for 1000 times navigation between A* and our algorithm.

Algorithms	Ave time cost (h)	Ave fuel cost
A*	1.8	$2.1 \times 10^{-2} L$
MKE_v3	1.1	$1.2 \times 10^{-2} L$

and experiment results show that our algorithm has better convergence speed and convergence accuracy. PSO variants have a fatal weakness that large population size did not improve the optimization results significantly, so is the weakness to optimize high dimension problems. Our algorithm makes better use of particles' cooperation, and it's fully demonstrated on large scale optimization. An application of urban area vehicle navigation

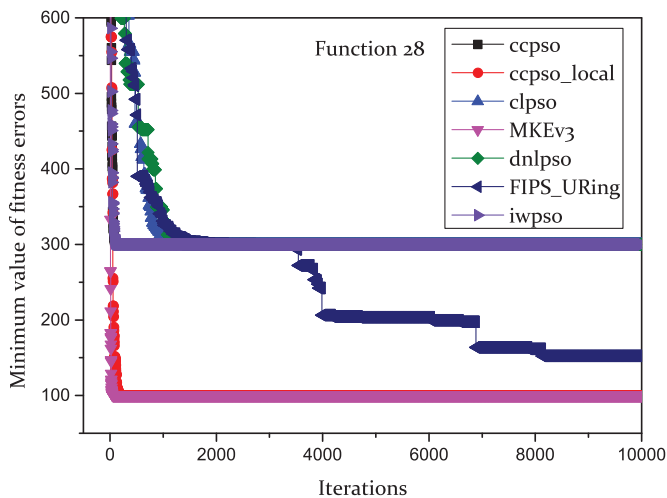


Fig. 15. Comparison of different algorithms on function 28.

within least fuel consumption is discussed and our algorithm gives the least fuel consumption navigation, and it outperforms A* and Dijkstra algorithm. Our algorithm can also be degraded into PSO form and Differential Evolution(DE) form. We also can reduce iterations/generations of particles' evolution by increasing the particle population size to achieve equal number of function evaluations, and it makes a better performance, the analysis will be discussed in the next paper.

References

- [1] J. Kennedy, R. Eberhart, Particle swarm optimization, *Proceedings of IEEE International Conference on Neural Networks* 4 (2) (1995) 1942–1948.
- [2] R.C. Eberhart, J. Kennedy, A new optimizer using particle swarm theory, in: *Proceedings of the Sixth International Symposium on Micro-Machine and Human Science*, vol. 1, 1995.
- [3] Y. Shi, R. Eberhart, A modified particle swarm optimizer, in: *Proceedings of the 1998 IEEE International Conference on Evolutionary Computation Proceedings, 1998 and IEEE World Congress on Computational Intelligence*, 1998.
- [4] R.C. Eberhart, Y. Shi, Comparing inertia weights and constriction factors in particle swarm optimization, in: *Proceedings of the 2000 Congress on Evolutionary Computation*, 2000, vol. 1, IEEE, 2000.
- [5] M. Clerc, J. Kennedy, The particle swarm-explosion, stability, and convergence in a multidimensional complex space, *IEEE Trans. Evol. Comput.* 6 (1) (2002) 58–73.
- [6] J. Kennedy, Small worlds and mega-minds: effects of neighborhood topology on particle swarm performance, in: *Proceedings of the 1999 Congress on Evolutionary Computation*, 1999, CEC 99, vol. 3, IEEE, 1999.
- [7] J. Kennedy, R. Mendes, Population Structure and Particle Swarm Performance, 2002.
- [8] P.N. Suganthan, Particle swarm optimiser with neighbourhood operator, in: *Proceedings of the 1999 Congress on Evolutionary Computation*, 1999, CEC 99, vol. 3, IEEE, 1999.
- [9] R. Mendes, J. Kennedy, J. Neves, Watch thy neighbor or how the swarm can learn from its environment, in: *Proceedings of the Swarm Intelligence Symposium*, 2003, SIS03, IEEE, 2003.
- [10] R. Mendes, Population Topologies and Their Influence in Particle Swarm Performance, Universidade do Minho, 2004. *Dissertation*.
- [11] J.J. Liang, P.N. Suganthan, Dynamic multi-swarm particle swarm optimizer, in: *Proceedings of IEEE Swarm Intelligence Symposium*, 2005, SIS 2005, IEEE, 2005.
- [12] J.J. Liang, et al., Comprehensive learning particle swarm optimizer for global optimization of multimodal functions, *IEEE Trans. Evol. Comput.* 10 (3) (2006) 281–295.
- [13] Md. Nasir, et al., A dynamic neighborhood learning based particle swarm optimizer for global numerical optimization, *Inf. Sci.* 209 (2012) 16–36.
- [14] J. Garcia-Nieto, A.C. Olivera, E. Alba, Optimal cycle program of traffic lights with particle swarm optimization, *IEEE Trans. Evol. Comput.* 17 (2013) 823–839.
- [15] M. Gong, et al., Complex network clustering by multiobjective discrete particle swarm optimization based on decomposition, *IEEE Trans. Evol. Comput.* 18 (2014) 82–97.
- [16] H. Wang, X. Yan, Optimizing the echo state network with a binary particle swarm optimization algorithm, *Knowl.-Based Syst.* 86 (2015) 182–193.
- [17] K.Y. Huang, I.-H. Li, A multi-attribute decision-making model for the robust classification of multiple inputs and outputs datasets with uncertainty, *Appl. Soft Comput.* 38 (2016) 176–189.
- [18] O. Matei, et al., An improved immigration memetic algorithm for solving the heterogeneous fixed fleet vehicle routing problem, *Neurocomputing* 150 (2015) 58–66.
- [19] P.E. Hart, N.J. Nilsson, B. Raphael, A formal basis for the heuristic determination of minimum cost paths, *IEEE Trans. Syst. Sci. Cybern.* 4 (2) (1968) 100–107.
- [20] W.E. Dijkstra, A note on two problems in connexion with graphs, *Numer. Math.* 1 (1) (1959) 269–271.
- [21] H. Nikolaus, S. Finck, R. Ros, A. Auger, Real-parameter Black-box Optimization Benchmarking 2009: Oiseless Functions Definitions, Research Report RR-6829, 2009. <inria-00362633>.
- [22] J.J. Liang, et al., Problem Definitions and Evaluation Criteria for the CEC 2013 Special Session on Real-parameter Optimization, Technical report 201212, Computational Intelligence Laboratory, Zhengzhou University, Zhengzhou, China, 2013. Nanyang Technological University, Singapore.
- [23] K. Tang, et al., Benchmark Functions for the CEC2008 Special Session and Competition on Large Scale Global Optimization, *Nature Inspired Computation and Applications Laboratory*, USTC, China, 2007, pp. 153–177.
- [24] F. Van den Bergh, A.P. Engelbrecht, A cooperative approach to particle swarm optimization, *IEEE Trans. Evol. Comput.* 8 (3) (2004) 225–239.
- [25] H.Y. Huang, P.E. Luo, M. Li, et al., Performance evaluation of SUVnet with real-time traffic data, *IEEE Trans. Veh. Technol.* 56 (2007) 3381–3396.
- [26] Z. Meng, J.S. Pan, A simple and accurate global optimizer for continuous spaces optimization, *Genetic and Evolutionary Computing*, Springer International Publishing, 2015, pp. 121–129.