Telecommun Syst (2016) 62:403–415
DOI 10.1007/s11235-015-0088-4

A new meta-heuristic ebb-tide-fish-inspired algorithm for traffic navigation

Zhenyu Meng[1] · **Jeng-Shyang Pan**[1,2] · **Abdulhameed Alelaiwi**[3]

Published online: 10 September 2015

Abstract More and more bio-inspired or meta-heuristic algorithms have been proposed to tackle the tough optimization problems. They all aim for tolerable velocity of convergence, a better precision, robustness, and performance. In this paper, we proposed a new algorithm, ebb tide fish algorithm (ETFA), which mainly focus on using simple but useful update scheme to evolve different solutions to achieve the global optima in the related tough optimization problem rather than PSO-like velocity parameter to achieve diversity at the expenses of slow convergence rate. The proposed ETFA achieves intensification and diversification in a new way. First, a flag is used to demonstrate the search status of each particle candidate. Second, the single search mode and population search mode tackle the intensification and diversification for tough optimization problem respectively. We also compare the proposed algorithm with other existing algorithms, including bat algorithm, cat swarm optimization, harmony search algorithm and particle swarm optimization. Simulation results demonstrate that the proposed ebb tide fish algorithm not only obtains a better precision but also gets a better convergence rate. Finally, the proposed algorithm is used in the application of vehicle route optimization in Intelligent Transportation Systems (ITS). Experiment results show that the proposed scheme also can be well performed for vehicle navigation with a better performance of the reduction of gasoline consumption than the shortest path algorithm (Dijkstra Algorithm) and A* algorithm.

Keywords Benchmark function · Bio-inspired algorithm · Ebb tide fish algorithm · Gasoline consumption · Meta-heuristic · optimization · Vehicle navigation

1 Introduction

There are many kinds of demands for tough optimization problems in our lives. To tackle these problems often begins by designing related models incorporating with corresponding constraints. Therefore, more and more meta-heuristic algorithms have been proposed, such as particle swarm optimization (PSO) [6,16,22,24], differential evolution [7,20,23] etc., and many evolutionary theories have been proposed for these approximations or optimizations [2,8,14,21]. PSO has undergone many changes, or in other words, many meta-heuristic algorithms can be simplified into PSO form for optimization such as bat algorithm [26], cuckoo search algorithm[27] and cat swarm optimization [4], etc. They are all becoming powerful methods for solving many tough optimization problems. Paying more attention to detail, we find that these bio-inspired or meta-heuristic algorithms are all derived, obviously simplified, from models of behaviors of biological system and physical systems in nature. How to understand the nature well, what models can be extracted for computational use and how the model works all determine the performance of the algorithm. Neural network [15]

✉ Jeng-Shyang Pan
jengshyangpan@gmail.com

Zhenyu Meng
mzy1314@gmail.com

[1] Innovative Information Industry Research Center, Department of Computer Science, Harbin Institute of Technology Shenzhen Graduate School, Shenzhen, China

[2] College of Information Science and Engineering, Fujian University of Technology, Fuzhou, China

[3] Department of Software Engineering, College of Computer & Information Sciences, King Saud University, P.O. Box 51178, Riyadh 11543, Kingdom of Saudi Arabia

is model of neurons, simulated annealing [1,18] involves heating and controlled cooling of a material, PSO is developed based on the swarm behavior of birds, bat algorithm and cuckoo search are inspired by bat and cuckoo respectively, harmony search [12,28] is derived from the improving process of composing a piece of music, and firefly algorithm is derived from the flashing behavior of fireflies. Each of these algorithms for optimization has its own advantages and disadvantages. For example, simulated annealing guarantees to find the optimal solution when the cooling process is slow enough and the simulation time is long enough. However, the adjustment in parameters does affect the convergence process [3,11,17,24,29]. PSO adaptation has been shown to successfully optimize a wide range of continuous functions, however, the model can be parameterized that the particle system consistently converges on local optima, except for a special class of functions, convergence on global optima cannot be proven [5].

The publication "no free lunch theorems for optimization" [25], claims that for any algorithm, any elevated performance over one class of problems is offset by performance over another class. The theorems result in a geometric interpretation of what it means for an algorithm to be well suited to optimization problems. In other words, if Algorithm A performs better than Algorithm B for some optimization functions, then B will outperform A for other functions, or if averaged over all possible function space, both algorithm A and B will perform on average equally well. It sounds very disappointing, but we do not need the average over all possible functions for a given optimization problem, and what we need is to find the best solutions rather than evaluating the average overall possible function space. So there are algorithms indeed outperform others for given types of optimization problems.

In this paper, we intent to propose a new meta-heuristic approach to optimization, the ebb tide fish algorithm. This algorithm is inspired by small fish in ebb tides. The capability of sensing vibration and flow is very fascinating and it also make possible preying or fleeing for lives. The rest of the paper is organized as follows. In Sect. 2, we formulate the prey of the ebb tide fish, the communication between fish, and we also give descriptions on how the algorithm works. We also make some literal analysis of the update scheme of ETFA and comparisons with respect to PSO and DE. In Sect. 3, we will introduce the benchmark functions that validate the proposed algorithm and give descriptions of the good performance of our proposed algorithm. In Sect. 4, we give the comparison with other algorithms for optimization. In Sect. 5, we will discuss an extension of the proposed algorithm, the application for route optimization. In Sect. 6, we will discuss the algorithm generally, the advantages and the disadvantages, conclusions of the paper and some future works.

2 Ebb tide fish

The small fishes in ebb tides are fascinating creatures. They have lateral line to make perception of the tide flow, sounds and the vibrations in water. They also use their ears to percept sounds and vibrations. They are very sensitive to low frequency sounds, especially in the frequencies from $6Hz$ to $16Hz$, while some other kinds of fishes have a wider range from $1Hz$ to $150Hz$. They also can make many kinds of sound by vibrating of swimming bladder or grinding of bones or teeth, to communicate with each other.

2.1 Behavior of ebb tide fish

When the fish is preying plankton or eating the weeds, it is often attracted by some other fishes in the population who have found a safer place with abundant food. Communications among the fishes lead to finding the safest place with abundant food, and staying in the crowd also improves the possibility to survive as it reduces the risk of being hunted as an individual. In addition, some single fish would like to search around alone rather than being attracted by the population. They also transmit their findings to other fish in the population by special sounds frequently. They swim around several times to find whether there is a better place for food. The searching range is often delimited by a certain searching radius.

2.2 Ebb tide fish algorithm

The speed of sound propagation in sea water is typically $V = 1450$ m/s, and the equation of transmission time is $t = \frac{2S}{V}$, so the time is much short than sound propagation in the air with the velocity $V = 340$ m/s. The quick response means that the fish has more time to do some response acts than creatures in the air. The food (plankton/weeds) of the fish is increasing with the depth increment in the model of our algorithm, so fish would like to swim deeper to search for food, and it make it possible to find the local optima of an optimization. The noise of ebb tide wave is decreased by depth, so the fish would also like a deep swim to escape from a potential danger in noisy surroundings. If a fish find the temporal global best location abundance of food, others will gather in. If the noise in a local area is larger than a default tolerance/threhold of the fish $A \geq A_{thres}$, the fish will escape and find a new place for prey. $A = A_0 * \frac{1}{2\pi} e^{-(h-h_0)^2}$, $h - h_0$ denotes the relative depth. Figure 1 shows the search behavior of fish.

For the ebb tide fish algorithm, we mainly focus on coordinate information rather than the velocity of the movement. The following equation Eq. 1 describes the position update scheme.

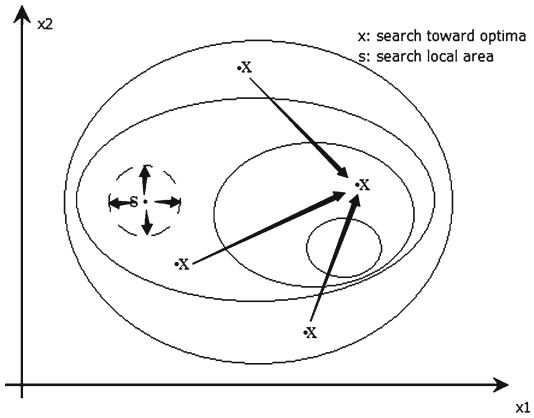


Fig. 1 The search behavior of the fish

$$X_i^{t+1} \leftarrow X_i^t + (X_{gbest}^t - X_i^t) * rand(); \tag{1}$$

When a single fish finds a safer place (which is less vibration and noise) with much more plankton/weeds, it would like to search the local area to find the optima location. The label $Flag = true$ denotes that the fish is a single search fish. The fish would also like to search a nearby place in diversified ways. So it will swim about for N times ($N = \frac{d_{count}}{2} * \beta$, and d_{count} denotes the number of dimensions in the searching domain. For simplification we use a fixed number $\beta = 5$.). As we know, only a small proportion of fish would like to search individually, others would like to forage following the population. So we set the proportion of the population for single swim search is $rate = 0.01$. Let $X_i = (x_1, x_2, ..., x_d)$ denotes the coordinate of the ith fish, $X_{center} = (x_{1_0}, x_{2_0}, ..., x_{d_0})$ denotes the searching domain center, and the local searching radius of each dimension is $r = \frac{1}{\beta} * (x_i - x_{i_0})$. In implementation, we first record the position of the single search fish, and then the fish randomly swims for a distance r either in the positive direction of one dimension ($x_i \leftarrow x_i + r$) or in the negative direction ($x_i \leftarrow x_i - r$). After swimming for N times, the fish finds the local optima and uses the location X_{lbest} to update the recorded location X_{pos_i}. The algorithm is shown below in Algorithm 1.

In order to examine the proposed algorithm in detail and make comparisons with other well-known algorithms, first we take PSO update scheme into consideration with the equation shown in Eq. 2.

$$\begin{cases} v_i^{t+1} \leftarrow v_i^t + c1 * (X_{fb} - x_i) + c2 * (X_{gb} - x_i); \\ X_i^{t+1} \leftarrow X_i^t + v_i^{t+1}; \end{cases} \tag{2}$$

We define $C = C1 + C2$, $p \leftarrow \frac{C1 * X_{fb} + C2 * X_{gb}}{C1 + C2}$, then Eq. 2 can be changed to Eq. 3.

$$\begin{cases} v(t+1) = v(t) + C * (p - x(t)) \\ x(t+1) = x(t) + v(t+1); \end{cases} \tag{3}$$

Algorithm 1 Pseudo code of the ebb tide fish algorithm

Require:
Initialize the searching space $V(v_1, v_2, ..., v_d)$ and the benchmark function $f(X)$(X denotes the coordinate of a virtual fish).
Ensure:
1: **while** $exeTime < MaxIteration$ **do**
2: **if** $exeTime = 1$ **then**
3: Generate the fish population T_j ($j = 1, 2, ..., n$) with coordinate $X_j = (x_1, x_2, ..., x_d)^T$ and generate single search fish and change its $Flag_j$.
4: **end if**
5: **if** $exeTime > 1$ **then**
6: **for** $pSize = 1 : PopSize$ **do**
7: **if** $Flag_i = true$ & $A_i < A_{thres}$ **then**
8: **for** $C_{swim} = 1 : N$ **do**
9: $x_i = x_i \pm r$
10: Calculate the benchmark value and record local best X_{lbest}^t.
11: **end for**
12: $X_{pos_i} \leftarrow X_{lbest}^t$
13: $Flag_i = false$
14: **else**
15: $X_{pos_i} \leftarrow X_i^t + (X_{gbest}^t - X_i^t) * rand();$
16: **end if**
17: Generate single search fish and change its $Flag$
18: **end for**
19: **end if**
20: Calculate the benchmark value of all the fish.
21: **if** $f(X_{pos_i})$ is optima rather than $f(X_i^t)$ **then**
22: $X_i^{t+1} \leftarrow X_{pos_i}$
23: **end if**
24: Record the optima fish with coordinate X_{gbest}^t.
25: **end while**
Output:
The global optima X_{gbest}.

Consequently, the form can also be changed to Eq. 4 by defining $y_t = p - x_t$.

$$\begin{cases} v_{t+1} \leftarrow v_t + C * y_t \\ y_{t+1} \leftarrow -v_t + (1 - C) * y_t; \end{cases} \tag{4}$$

It can also be written in the matrix style shown in Eq. 5.

$$\begin{bmatrix} v_{t+1} \\ y_{t+1} \end{bmatrix} = \begin{bmatrix} 1 & C \\ -1 & 1 - C \end{bmatrix} * \begin{bmatrix} v_t \\ y_t \end{bmatrix} \tag{5}$$

When $t = 0$, $v_0 = 0$, so y_{t+1} and v_{t+1} can be calculated by Eq. 6 through y_1 and v_1.

$$\begin{bmatrix} v_{t+1} \\ y_{t+1} \end{bmatrix} = P * \begin{bmatrix} e_1 & 0 \\ 0 & e_2 \end{bmatrix}^t * P^T * \begin{bmatrix} v_1 \\ y_1 \end{bmatrix} \tag{6}$$

We put the two kind update factors into Eq. 7 (the ceil one is update factor of PSO, the bottom one is of ETFA) to make a transparent comparison. From the update factor, we can see that the velocity parameter results in diversity solutions in recompense for bad convergence rate.

$$\begin{cases} f_{pso} = A * v_1 - B * x_1 \\ f_{etfa} = C - D * x_1; \end{cases} \tag{7}$$

Then we take differential evolution (DE) algorithm into consideration. Eq. 8 shows the mutation scheme that the new candidate is a single dimension based combination of the mutation candidate and the original candidate before mutation.

$$v_{i,G+1} = x_{r_1,G} + F * (x_{r_2,G} - x_{r_3,G}) \tag{8}$$

Eq. 9 shows how to generate the final candidate of DE. The detailed description of DE algorithm is discussed in [23].

$$u_{ji,G+1} = \begin{cases} v_{ji,G+1}, & if(randb(j) \leq CR) or j = rnbr(i) \\ x_{ji,G}, & if(randb(j) > CR) or j = rnbr(i) \end{cases} \tag{9}$$

Dimension based update scheme of DE achieved intensity and diversity for optimization, and it has been proved to be a state of art algorithm. The proposed algorithm in this paper is vector based update scheme, and we add local search to fulfill diversity for optimization. From algorithm complexity perspective of view, ETFA is less time-consumption and detailed description and comparison will be illustration in another paper which focus on the state of art and comparison with DE.

3 Performance evaluation

There are many benchmark functions for the validation of new algorithm. Take the Ackley 2-dimension function for example (shown in Fig. 2), we get the global optima 0 at $X_j = (0, 0)^T$, X_j is a column vector. To make a more accurate validation, we use many benchmark functions, listed in Table 1, to test our proposed algorithm. The equations of the functions and the minimum values in search domain V are given in Table 1 and Table 2 respectively. In our implementation, we use 50 different populations of virtual fish, with 30 virtual fish in each population to tackle the optimization problem, and we run 1,000 times and the average convergence curves are also calculated to make a comparison with other different algorithms. Figure 3 shows the convergence curve of the proposed algorithm with the benchmark function – Ackley function. For simplification, A_{thres} is set a big enough value so that equation $A_i < A_{thres}$ is always true in our simulation, and we used all of the benchmark functions listed in Table 1 for the validation of our proposed algorithm, and the simulation results show that our algorithm performs very well. Only a randomly selected 6 figures (Figs. 3, 4, 5, 6, 7, 8) are shown here in consideration of the total length of the paper, but we will give full comparisons of different optimization algorithms using all the benchmark functions in the later part of the paper.

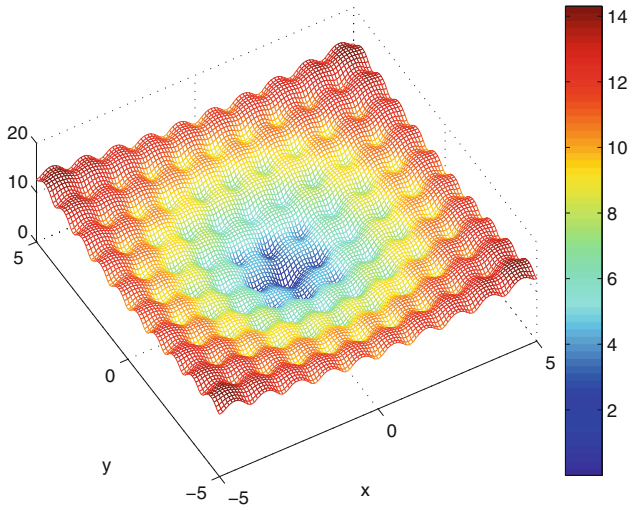


Fig. 2 Ackley mesh

4 Comparison

We make as much comparisons as possible to test the performance of our proposed algorithm against other optimization algorithms, such as bat algorithm, cat swarm optimization, harmony search, and particle swarm optimization. The parameter we use for the comparison is 10 populations of virtual fish and for each population there are 30 virtual units ($pops = 10$, $popsize = 30$). The average curves of all the populations are calculated and shown for comparisons of all the algorithms in the figures.

There are mainly four aspects for the evaluation of optimization algorithms, velocity of convergence, precision, robustness and performance, and there also are two obviously approaches for the evaluation in experiments: to compare the numbers of function evaluations for a given tolerance or accuracy, or to compare their accuracies for a fixed number of function evaluations [19]. In this paper, we use the second criteria for experiments. We run 150 iterations ($exetime = 150$) for each population and make comparison of the convergence velocity and precision. The 10 different populations, with each population 150 iteration times, make a statistical analysis.

In our implementation for the bat algorithm, the default parameters A_i^0 in $[1, 2]$, $A_{min} = 0$ (or $A_i^0 = 100$, $A_{min} = 1$), $\alpha = \gamma = 0.9$, r_i^0 around 0, do not have a good convergence rate (there is no curve can be seen in comparison figures during the first 150 iteration times for different benchmark functions, as after 150 iterations the fitness value is

Table 1 Benchmark functions

No.	Name	Benchmark function
1	Ackley	$f(X_j) = -20 * e^{-0.2*\sqrt{\frac{1}{d}*\sum x_i^2}} - e^{\frac{1}{d}*\sum cos(2*\pi*x_i)} + 20 + e$
2	Beale	$f(x, y) = (1.5 - x + x * y)^2 + (2.25 - x + x * y^2)^2 + (2.625 - x + x * y^3)^2$
3	Booth	$f(x, y) = (x + 2y - 7)^2 + (2x + y - 5)^2$
4	CrossInTray	$f(x, y) = -0.0001(\lvert \sin(x) * sin(y) * e^{\left\lvert 100 - \frac{\sqrt{x^2+y^2}}{\pi} \right\rvert} \rvert + 1)^{0.1}$
5	Easom	$f(x, y) = -\cos(x) * \cos(y) * e^{-((x-\pi)^2+(y-\pi)^2)}$
6	Eggholder	$f(x, y) = -(x + 47) * sin(\sqrt{\left\lvert y + \frac{x}{2} + 47 \right\rvert} - x * \sin(\sqrt{\lvert x - (y + 47) \rvert})$
7	Goldstein	$f(x, y) = (1 + (x + y + 1)^2 * (19 - 14x + 3x^2 - 14y + 6xy + 3y^2)) * (30 + (2x - 3y)^2 * (18 - 32x + 12x^2 + 48y - 36xy + 27y^2))$
8	HolderTable	$f(x, y) = -\left\lvert \sin(x) * \cos(y) * e^{\left\lvert 1 - \frac{\sqrt{x^2+y^2}}{\pi} \right\rvert} \right\rvert$
9	Levi	$f(x, y) = \sin^2(3\pi x) + (x - 1)^2 * (1 + \sin^2(3\pi y)) + (y - 1)^2 * (1 + \sin^2(2\pi y))$
10	Matyas	$f(x, y) = 0.26(x^2 + y^2) - 0.48xy$
11	McCormick	$f(x, y) = sin(x + y) + (x - y)^2 - 1.5x + 2.5y + 1$
12	Rosenbrock	$f(X_j) = \sum_{i=1}^{d-1} (100(x_{i+1} - x_i^2)^2 + (x_i - 1)^2)$
13	Schaffer	$f(x, y) = 0.5 + \frac{\sin^2(x^2 - y^2) - 0.5}{(1 + 0.001(x^2 + y^2))^2}$
14	Sphere	$f(X_j) = \sum_{i=1}^{d} x_i^2$
15	Styblinski	$f(X_j) = \frac{\sum_{i=1}^{n} x_i^4 - 16x_i^2 + 5x_i}{2}$
16	ThreeHumpCamel	$f(x, y) = 2x^2 - 1.05x^4 + \frac{x^6}{6} + xy + y^2$

Table 2 Search domain and minimum of benchmark functions

No.	Name	Search domain	Minimum value
1	Ackley	$-5 \le x, y \le 5$	$f(0, 0) = 0$
2	Beale	$-4.5 \le x, y \le 4.5$	$f(3, 0.5) = 0$
3	Booth	$-10 \le x, y \le 10$	$f(1, 3) = 0$
4	CrossInTray	$-10 \le x, y \le 10$	$f(\pm 1.34941, \pm 1.34941) = -2.06261$
5	Easom	$-100 \le x, y \le 100$	$f(\pi, \pi) = -1$
6	Eggholder	$-512 \le x, y \le 512$	$f(512, 404.2319) = -959.6407$
7	Goldstein	$-2 \le x, y \le 2$	$f(0, -1) = 3$
8	HolderTable	$-10 \le x, y \le 10$	$f(\pm 8.05502, \pm 9.66459) = -19.2085$
9	Levi	$-10 \le x, y \le 10$	$f(1, 1) = 0$
10	Matyas	$-10 \le x, y \le 10$	$f(0, 0) = 0$
11	McCormick	$-1.5 \le x \le 4, -3 \le y \le 4$	$f(-0.54719, -1.54719) = -1.9133$
12	Rosenbrock	$-5 \le x, y \le 5$	$f(1, 1, ..., 1) = 0$
13	Schaffer	$-100 \le x, y \le 100$	$f(0, 0) = 0$
14	Sphere	$-5 \le x, y \le 5$	$f(0, 0, ..., 0) = 0$
15	Styblinski	$-5 \le x, y \le 5$	$f(-2.903534, ..., -2.903534) = -39.16599 * 2$
16	ThreeHumpCamel	$-5 \le x, y \le 5$	$f(0, 0) = 0$

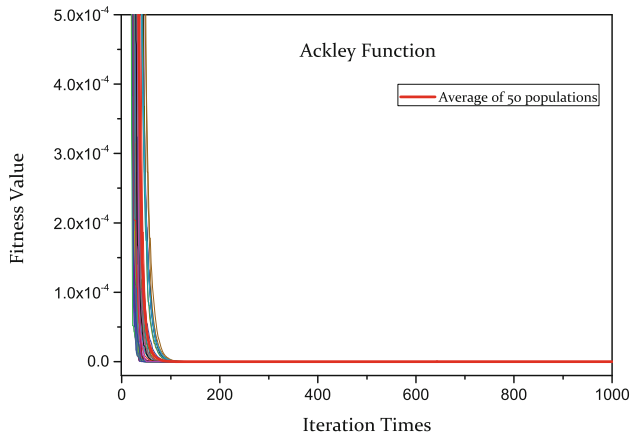


Fig. 3 Ackley function

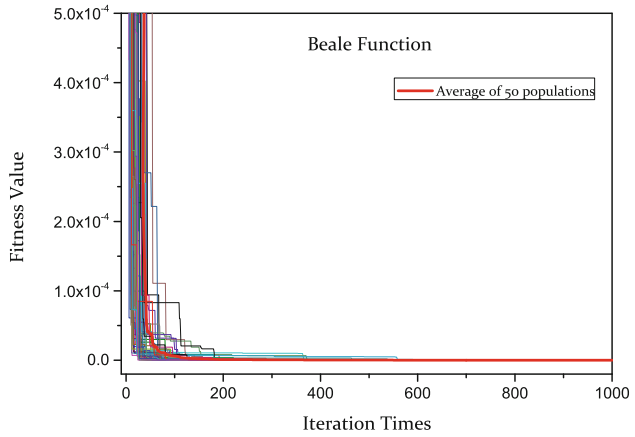


Fig. 4 Beale function

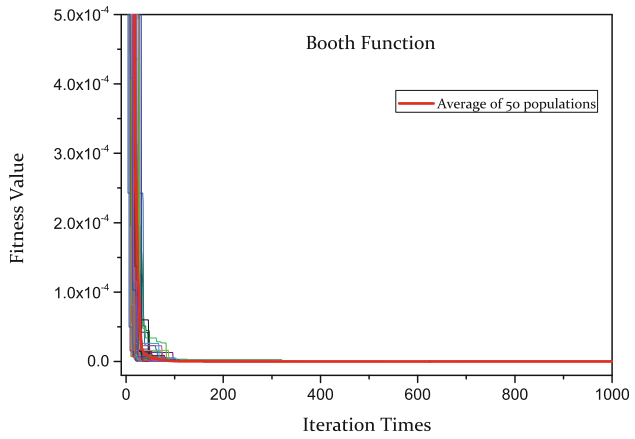


Fig. 5 Boooth function

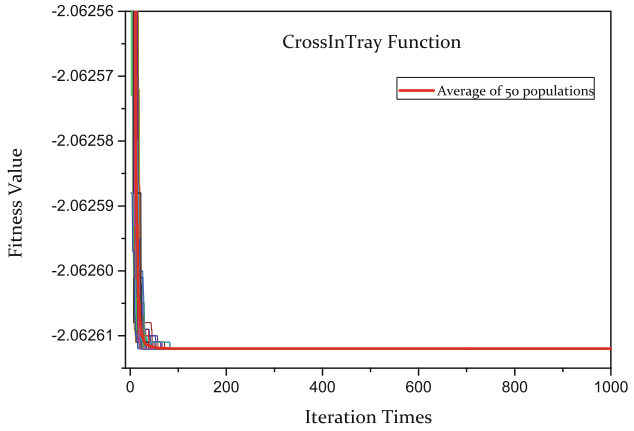


Fig. 6 CrossInTray function

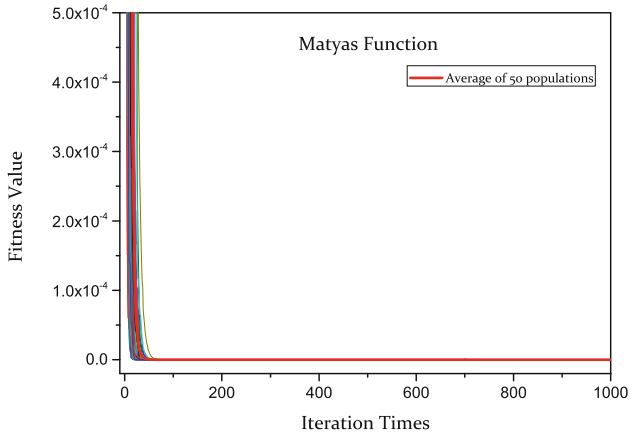


Fig. 7 Matyas function

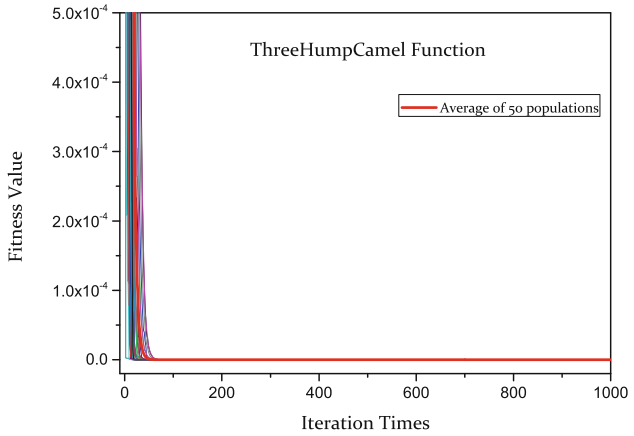


Fig. 8 ThreeHumpCamel function

still too large.), so we use the parameter $A_i = 0$, $r_i = 1$, $\alpha = \gamma = 0.9$, the bat algorithm in particle swarm optimization form (BA(pso)) and the parameter $A_i = 0.7$, $r_i = 0.7$, $\alpha = \gamma = 0.9$, the bat algorithm in harmony search form (BA(hs)), to make comparisons in Figures. For cat swarm optimization, we use the parameter max velocity $v_{max} = 0.3$, the average velocity $v_{ave} = 0.1$, the tracing rate $r = 0.02$, the number of seeking pool $smp = 5$, the inertial coefficient $c = 2$. For the particle swarm optimization, we use standard version for the comparison, the velocity inertial coefficient $c = 1$, $c_1 = c_2 = 2$. Figure 9 shows the comparison of our proposed algorithm, ebb tide fish algorithm, with the

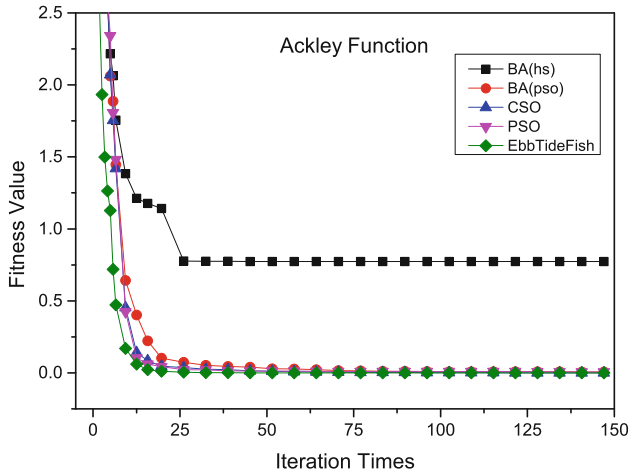


Fig. 9 Comparison of benchmark function—Ackley

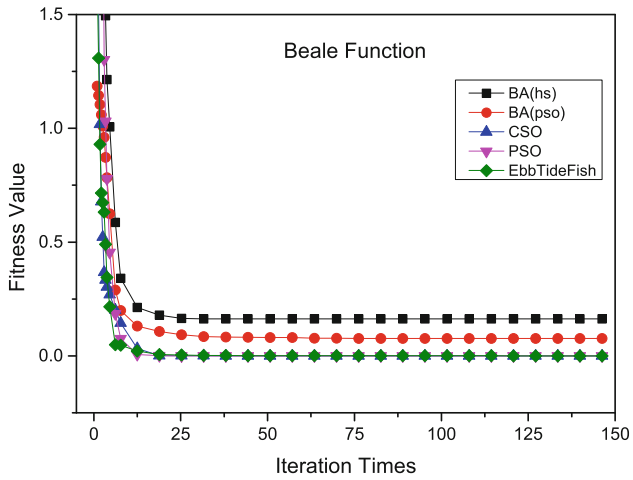


Fig. 10 Comparison of benchmark function—Beale

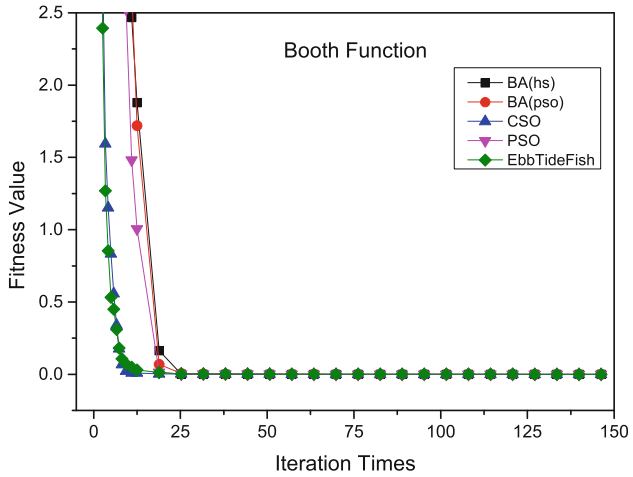


Fig. 11 Comparison of benchmark function—Booth

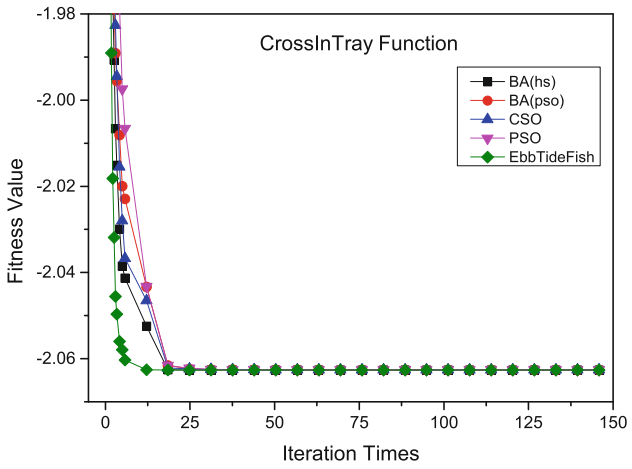


Fig. 12 Comparison of benchmark function—CrossInTray

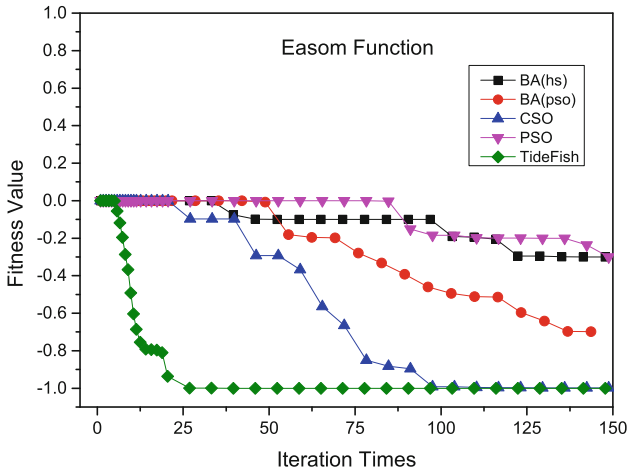


Fig. 13 Comparison of benchmark function—Easom

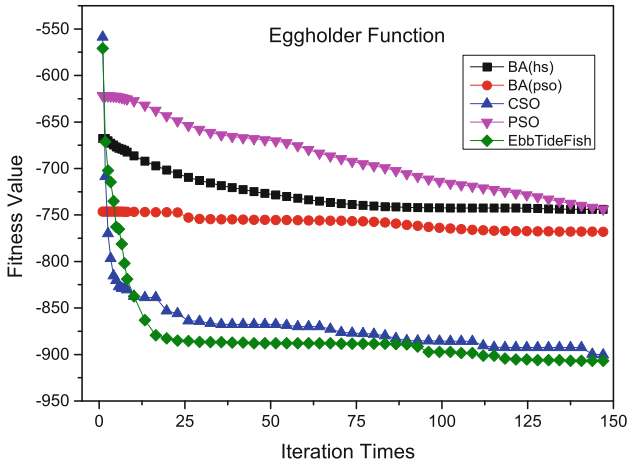


Fig. 14 Comparison of benchmark function—Eggholder

other algorithms for Ackley function. Figure 10 shows the comparison for Beale function. Figures 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24 show the comparisons for the rest of the benchmark functions listed in Table 1. The optimization results of 150 iterations are shown in Table 3.

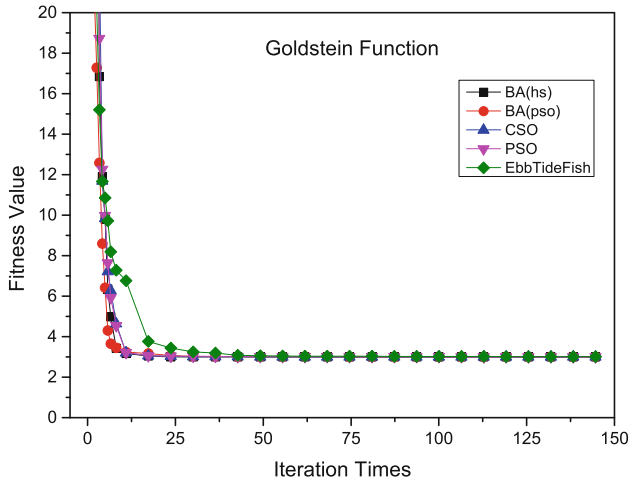


Fig. 15 Comparison of benchmark function—GoldStein

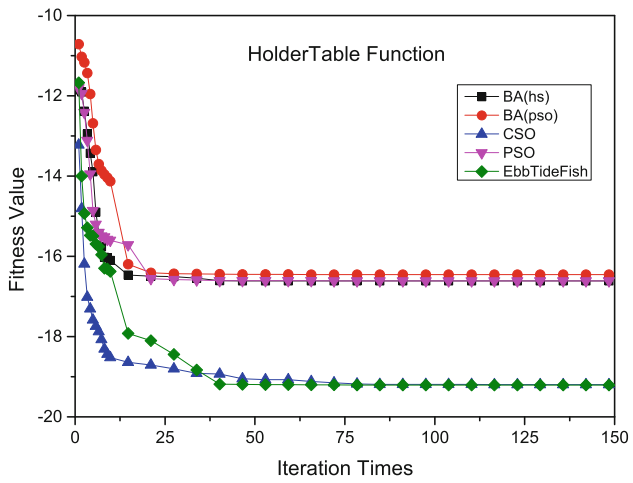


Fig. 16 Comparison of benchmark function—HolderTable

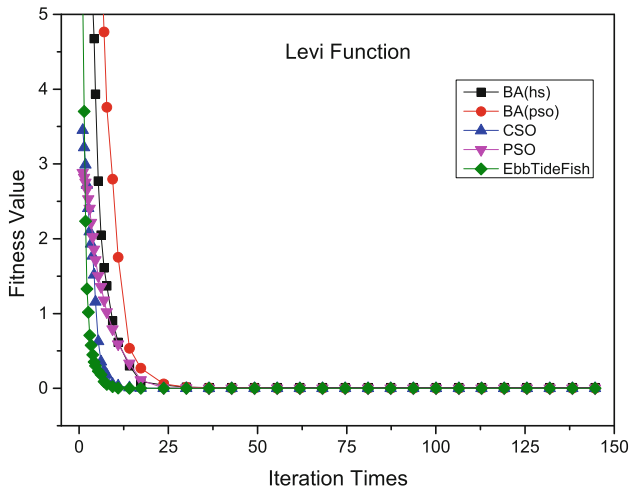


Fig. 17 Comparison of benchmark function—Levi

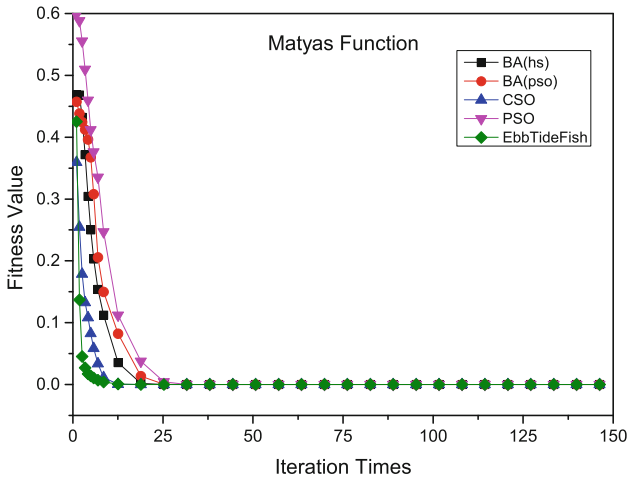


Fig. 18 Comparison of benchmark function—Matyas

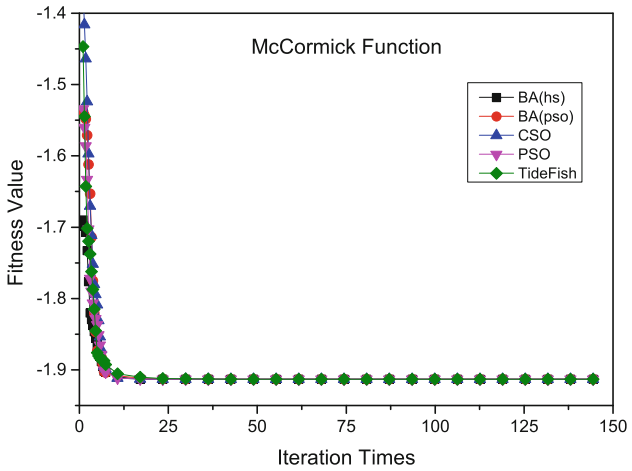


Fig. 19 Comparison of benchmark function—McCormick

5 Application of ebb-tide-fish algorithm

The proposed algorithm can be applied to solve diverse optimization problems with different applications [10]. In this paper, we will show the application of ETFA for vehicle gasoline consumption optimization in Intelligent Transportation Systems (ITS). In order to show the suitability and efficiency of the algorithm, an 8×8 grid network are employed for simulation. Ten thousand cars are mapped randomly into the grid network, so the density of traffic on the road can be simulated. In our simulation, the distance between two nodes in Fig. 25 is 2 km. After 10 thousand cars mapped into the grid, there are 89 cars on each road with the average about 22.4 m a car. The maximum velocity of the road is 72 km/h. The response time of a driver for emergency is 0.75s, so the safe distance is set to 20 m. We set two criteria for traffic congestion, one is the density becomes 1.2 times than normal situation, and the other is more than 3 car on 22.4 m. The

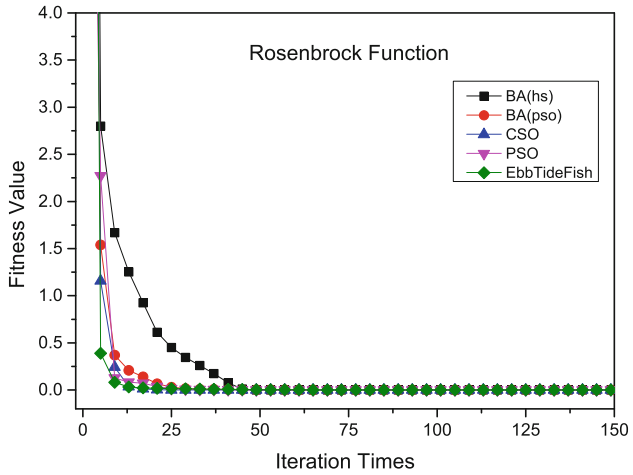


Fig. 20 Comparison of benchmark function—Rosenbrock

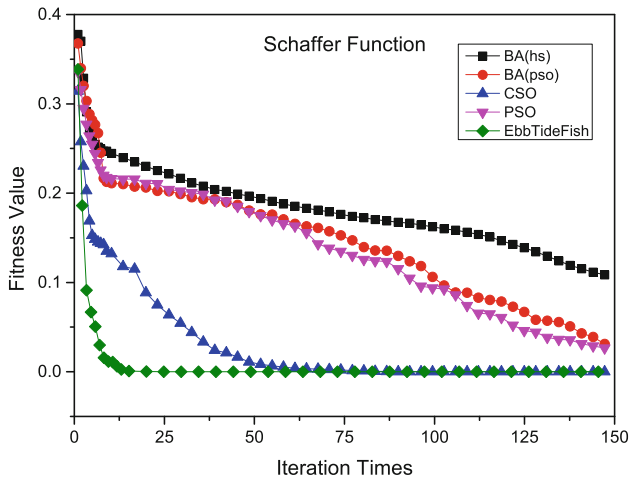


Fig. 21 Comparison of benchmark function—Schaffer

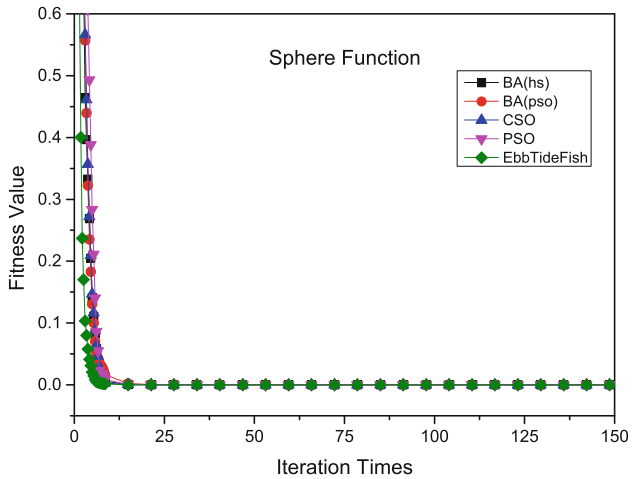


Fig. 22 Comparison of benchmark function—Sphere

red node in Fig. 25 shows the congestion node of one run of the simulation. The pink node is the start node and the yellow node is the destination node. The main objective of the

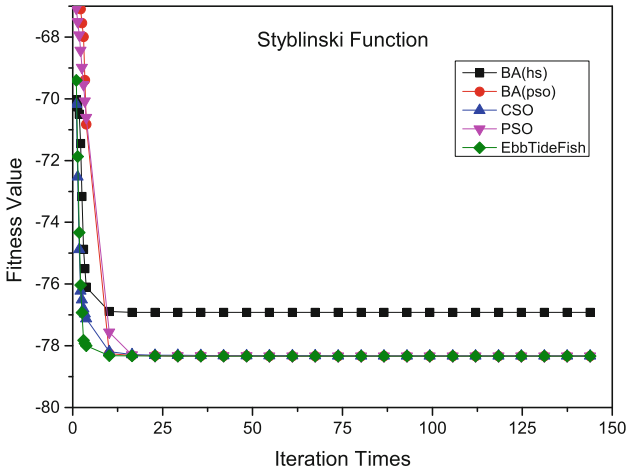


Fig. 23 Comparison of benchmark function—Styblinski

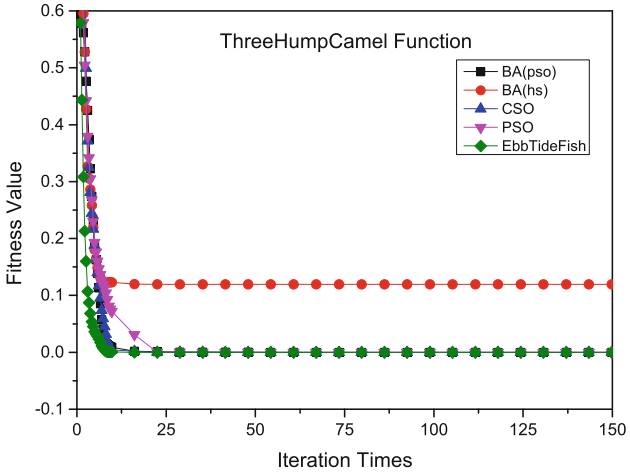


Fig. 24 Comparison of benchmark function—ThreeHumpCamel

paper is to make a navigation from the start to the destination within least gasoline cost.

5.1 Vehicle velocity and gasoline consumption

Recent research shows that there is an optimum velocity range for each car. Typical small gasoline ($< 1400\,\text{cm}^3$) Euro 4 passenger car is made as an example and Fig. 26[1] shows the optimal speed for minimum fuel consumption. From the chart, we can separate four areas with the corresponding to 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 continuous start and stop motion, and the traffic situation of in this range is often in traffic congestion. The second range is 30–55 kph, and the velocity range is common in sun-urban or rural areas. The third range is 55–80 kph, and this is the optimum velocity range that minimize

[1] http://www.myengineeringworld.net/2012/05/optimal-speed-for-minimum-fuel.html.

Table 3 Optimization results of benchmark functions after 150 iterations

No.	Name	Minimum value	BA(hs)	BA(pso)	CSO	PSO	Our algorithm
1	Ackley	0	7.7E−1	6.0E−3	1.8E−4	6.0E−3	**1.2E−8**
2	Beale	0	1.6E−1	7.6E−2	**2.0E−5**	2.5E−5	1.3E−4
3	Booth	0	**2.0E−9**	1.4E−5	6.0E−5	2.3E−5	1.6E−4
4	CrossInTray	−2.062612	−2.06261	−2.06261	−2.06261	−2.06261	**−2.062612**
5	Easom	−1	−1+6.0E−1	−1+3.0E−1	−1+2.0E−3	−1+7.0E−1	**−1+9.0E−7**
6	Eggholder	−959.6407	−744.0175	−768.1944	−900.5568	−747.8127	**−906.6949**
7	Goldstein	3	3	3+8.2E−4	**3+7.8E−4**	3+3.4E−3	3+6.0E−3
8	HolderTable	−19.2085	−16.6100	−16.4530	−19.1999	−16.6102	**−19.2071**
9	Levi	0	1.1E−2	1.2E−4	4.2E−5	9.6E−5	**3.4E−15**
10	Matyas	0	6.0E−11	2.7E−7	1.3E−9	2.4E−6	**4.0E−16**
11	McCormick	−1.913223	−1.9132	−1.9132	−1.9132	−1.9132	**−1.913223**
12	Rosenbrock	0	**5.2E−9**	2.2E−4	2.2E−4	2.0E−4	1.0E−1
13	Schaffer	0	1.0E−1	3.0E−2	1.9E−8	2.2E−2	**9.4E−15**
14	Sphere	0	3.8E−10	3.4E−6	2.1E−8	7.9E−6	**7.7E−22**
15	Styblinski	−78.3320	−76.9100	−78.3300	−78.3298	**−78.3320**	**−78.3320**
16	ThreeHumpCamel	0	8.3E−6	1.2E−1	1.6E−8	1.5E−5	**1.4E−21**

The best optimization result of each benchmark function are highlighted in bold

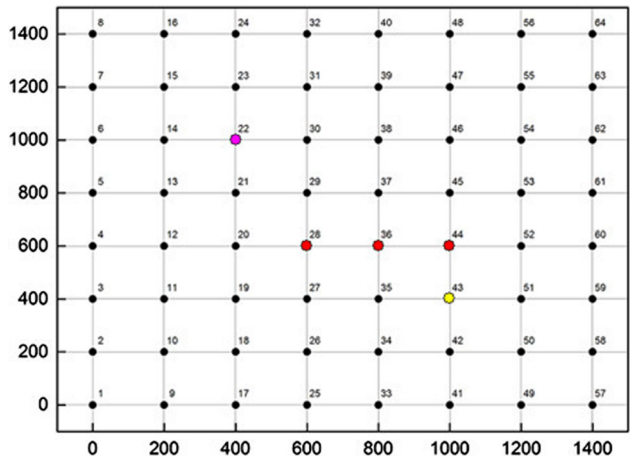


Fig. 25 Grid network of road simulation

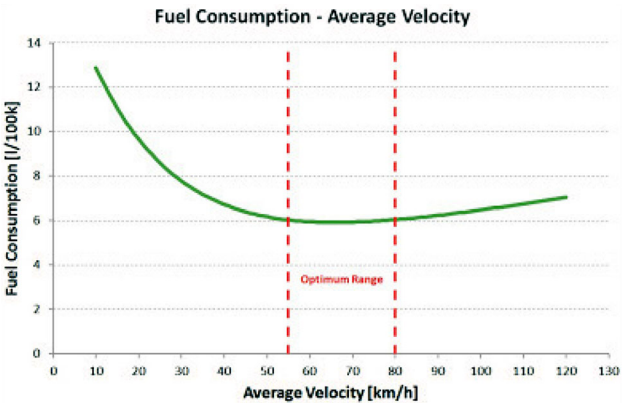


Fig. 26 Optimal speed for minimum fuel consumption

the fuel consumption. The last range is 80–120 kph, and the fuel consumption augments with the velocity increase. In our simulation, we choose one sample velocity in each range to make a simple gasoline consumption analysis. Table 4 shows the sample velocity in each range.

Take traffic status in Fig. 25 for example, the shortest path from the start point (Point 22) to destination point (Point 43) are shown as follows:

Path1: 22− > 30− > 38− > 46− > 45− > 44− > 43
Path2: 22− > 30− > 38− > 37− > 45− > 44− > 43
Path3: 22− > 30− > 38− > 37− > 36− > 44− > 43
Path4: 22− > 30− > 38− > 37− > 36− > 35− > 43
...
Path 20: 22− > 21− > 20− > 19− > 27− > 35− > 43

Table 4 Samples of the four ranges with velocity and fuel consumption

No.	Range	Velocity	Fuel consumption
1	0–30 kph	10 kph	12.5L/100 km
2	30–55 kph	36 kph	7 L/100 km
3	55–80 kph	72 kph	6 L/100 km
4	80–120 kph	108 kph	6.8 L/100 km

This driving paths are parts of the sequences which contains all nodes of the simulation grid. Each path from the start to the destination can be extended to a whole sequence with all the nodes of the grid simulation, and the weight of the nodes out of the path is set to zero.

5.2 Hamming distance of different sequences

The Hamming distance, in information theory, between two strings of equal length is the number of positions at which the corresponding symbols are different. It also measures the minimum number of substitutions required to change one string into the other, or the minimum number of errors that could have transformed one string into the other [13]. The consequent intersection nodes in vehicle navigation is firstly initialized and the fitness value is the gasoline consumption. The velocity of different congestion node are classified into four levels shown in Table 4. The distance between each travelling node sequence can be calculated by Hamming Distance with the equation Eq. 10. $d(X_{gbest}, X_i)$ is the minimum number of substitutions required to change so that X_i can be transformed to X_{gbest}.

$$d(X_{gbest}, X_i) = \sum_{j=1}^{d} x_{gbest_j} \oplus x_j \tag{10}$$

5.3 ETFA update scheme in path optimization

As we mentioned above, each fish is initialized a travelling sequence of the nodes, and the benchmark function is the gasoline consumption from the start to the destination. Of course, the congestion nodes within the sequence make the driving to consume more gasoline. After calculation, we can locate the temporal global optima and the following two moving schemes are "moving toward global optima" and "moving as a single search fish". In the first mode, Eq. 1 can be illustrated that X_i^t is updated by X_i^{t+1}, which has a shorter hamming distance, shown in Eq. 11, to the global optima. n denotes the number of nodes in the travelling sequence in Eq. 11. After we got the distance, d_{num} randomly selected positions in the sequence, from the first to the end of the travelling sequence, are assigned to the corresponding values of X_{gbest}. d_{num} can be calculated by Eq. 12. In the second mode, we just exchange two positions in the travelling sequence so that it can implement the local search of a single fish.

$$\begin{cases} d(X_i^{t+1}, X_{gbest}) = rand() * d_{small} \\ d_{small} = min\{d(X_i^t, X_{gbest}), n - d(X_i^t, X_{gbest})\} \end{cases} \tag{11}$$

$$d_{num} = d(X_i^t, X_{gbest}) - d(X_i^{t+1}, X_{gbest}) \tag{12}$$

In our simulation, ten thousand vehicles are mapped onto the grid network to simulate the traffic simulation. We make one thousand navigation with different start and destination and collect the distance of the journal, the velocity, travel time and gasoline consumption. Figure 27 shows the comparison of 1,000 times navigation between shortest path algorithm [9] and ETFA optimization and Fig. 28 shows

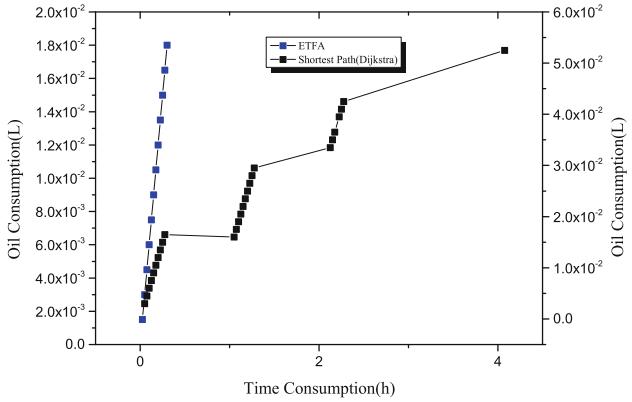


Fig. 27 Comparison with shortest path algorithm

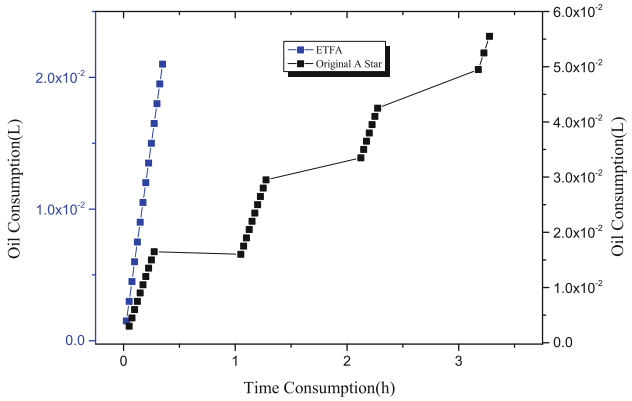


Fig. 28 Comparison with original A* algorithm

Table 5 The comparison of average fuel consumption and time consumption for 1,000 times navigation between Dijkstra and our algorithm

Algorithms	Ave time cost	Ave fuel cost
Dijkstra	1.2 h	1.1×10^{-2} L
Ebb-tide-fish	0.4 h	9.8×10^{-3} L

Table 6 The comparison of average fuel consumption and time consumption for 1,000 times navigation between A* and our algorithm

Algorithms	Ave time cost	Ave fuel cost
A*	1.8 h	2.1×10^{-2} L
Ebb-tide-fish	0.5 h	1.2×10^{-2} L

comparison of 1,000 times navigation between the proposed algorithm and the original A star algorithm [14]. Table 6 and Table 5 show the average fuel consumption and time consumption of 1,000 times navigation of A star algorithm and shortest path algorithm separately by comparison with our algorithm respectively. We can see that the proposed algo-

rithm outperforms on gasoline consumption over the two algorithms.

6 Conclusion

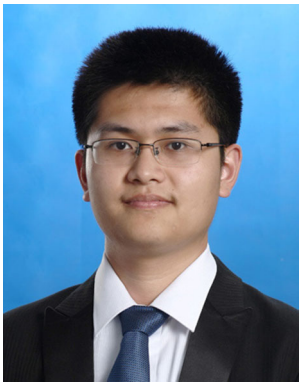
In this paper, we have formulated a new bio-inspired/meta-heuristic algorithm, the ebb tide fish algorithm, based on the behavior of the fishes in ebb tide. The proposed algorithm has been validated and compared with different optimization algorithms including bat algorithm, harmony search algorithm, cat swarm optimization and particle swarm optimization. Our proposed algorithm shows very good performance for many different benchmark functions. Our algorithm achieves a good convergence rate, make a whole simplification of particle swarm optimization, but do not result in over-convergence to some local optima by single search of the tide fish. As we know, the two major components of any meta-heuristic/bio-inspired algorithms are intensification and diversification, or exploitation and exploration. The more simple to implement the two components, the better the algorithm will be. Our proposed algorithm has less parameters and it avoids the tuning parameters problems for different/special problems. There are also some disadvantages of the proposed algorithm, such as the risk of convergence to local optima when the population is limited, the proof of convergence has not been down, etc.. We also give an example of application using the proposed algorithm in the paper. The application is the optimization of vehicle gasoline consumption in grid network. The performance shows that the proposed algorithm also well performed for path optimization with least gasoline consumption. An interesting future work will be the detailed discussion on the trajectory of the fish and other improvement of the algorithm.

Acknowledgments The authors extend their appreciation to the Deanship of Scientific Research at King Saud University, Riyadh, Saudi Arabia for funding this work through the research group project No. RGP-VPP-318.

References

1. Aarts, E., & Korst, J. (1988). Simulated annealing and Boltzmann machines: A stochastic approach to combinatorial optimization and neural computing.
2. Bäck, T., & Schwefel, H. P. (1993). An overview of evolutionary algorithms for parameter optimization. *Evolutionary Computation*, *1*(1), 1–23.
3. Bonabeau, E., Dorigo, M., & Theraulaz, G. (1999). *Swarm intelligence: From natural to artificial systems (No. 1)*. New York, NY: Oxford University Press.
4. Chu, S. C., Tsai, P. W., & Pan, J. S. (2006). Cat swarm optimization. In *PRICAI 2006: Trends in artificial intelligence* (pp. 854–858). Berlin: Springer.
5. Clerc, M., & Kennedy, J. (2002). The particle swarm-explosion, stability, and convergence in a multidimensional complex space. *IEEE Transactions on Evolutionary Computation*, *6*(1), 58–73.
6. Cui, Z., & Cai, X. (2009). Integral particle swarm optimization with dispersed accelerator information. *Fundamenta Informaticae*, *95*(4), 427–447.
7. Das, S., & Suganthan, P. N. (2011). Differential evolution: A survey of the state-of-the-art. *IEEE Transactions on Evolutionary Computation*, *15*(1), 4–31.
8. Deb, K. (2014). Multi-objective optimization. In: *Search methodologies* (pp. 403-449). New York: Springer.
9. Dijkstra, E.W. (1959). A note on two problems in connexion with graphs. *Numerische mathematik, 1*(1)
10. Eksioglu, B., Vural, A. V., & Reisman, A. (2009). The vehicle routing problem: A taxonomic review. *Computers & Industrial Engineering*, *57*(4), 1472–1483.
11. Engelbrecht, A. P. (2006). *Fundamentals of computational swarm intelligence*. New York: Wiley.
12. Geem, Z. W., Kim, J. H., & Loganathan, G. V. (2001). A new heuristic optimization algorithm: Harmony search. *Simulation*, *76*(2), 60–68.
13. Hamming, R. W. (1950). Error detecting and error correcting codes. *Bell System Technical Journal*, *29*(2), 147–160.
14. Hart, P. E., Nilsson, N. J., & Raphael, B. (1968). A formal basis for the heuristic determination of minimum cost paths. *IEEE Transactions on Systems Science and Cybernetics*, *4*(2), 100–107.
15. Hornik, K., Stinchcombe, M., & White, H. (1989). Multilayer feedforward networks are universal approximators. *Neural Networks*, *2*(5), 359–366.
16. Kennedy, J., & Eberhart, R. (1995). Particle swarm optimization. In: *Proceedings of IEEE international conference on neural networks* (Vol. 4, No. 2, pp. 1942–1948).
17. Kennedy, J., Kennedy, J. F., & Eberhart, R. C. (2001). *Swarm intelligence*. San Francisco: Morgan Kaufmann.
18. Kirkpatrick, S., Gelatt, C. D., & Vecchi, M. P. (1983). Optimization by simmulated annealing. *Science*, *220*(4598), 671–680.
19. Meng, Z., & Pan, J. S. (2015). A simple and accurate global optimizer for continuous spaces optimization. In *Genetic and evolutionary computing* (pp. 121–129). Springer International Publishing.
20. Parpinelli, R. S., & Lopes, H. S. (2011). New inspirations in swarm intelligence: A survey. *International Journal of Bio-Inspired Computation*, *3*(1), 1–16.
21. Polak, E. (1997). *Optimization: Algorithms and consistent approximations*. New York: Springer-Verlag.
22. Poli, R., Kennedy, J., & Blackwell, T. (2007). Particle swarm optimization. *Swarm Intelligence*, *1*(1), 33–57.
23. Storn, R., & Price, K. (1997). Differential evolution—A simple and efficient heuristic for global optimization over continuous spaces. *Journal of Global Optimization*, *11*(4), 341–359.
24. Van den Bergh, F., & Engelbrecht, A. P. (2006). A study of particle swarm optimization particle trajectories. *Information Sciences*, *176*(8), 937–971.
25. Wolpert, D. H., & Macready, W. G. (1997). No free lunch theorems for optimization. *IEEE Transactions on Evolutionary Computation*, *1*(1), 67–82.
26. Yang, X. S. (2010). A new metaheuristic bat-inspired algorithm. In *Nature inspired cooperative strategies for optimization (NICSO 2010)* (pp. 65-74). Berlin: Springer.
27. Yang, X. S., & Deb, S. (2009, December). Cuckoo search via Lvy flights. In: *World congress on nature biologically inspired computing, 2009. NaBIC 2009* (pp. 210-214). IEEE.
28. Yang, X. S. (2009). Harmony search as a metaheuristic algorithm. In *Music-inspired harmony search algorithm* (pp. 1–14). Berlin: Springer.

29. Yang, X. S., Cui, Z., Xiao, R., Gandomi, A. H., & Karamanoglu, M. (Eds.). (2013). *Swarm intelligence and bio-inspired computation: theory and applications*. Newnes.



Zhenyu Meng received the B. S. degree in Computer Science, Shandong Normal University in 2008, and the master degree in Computer Science from Harbin Institute of Technology in 2011. After the graduation of his master degree, he worked as a research assistant in Guangzhou Institute of Advanced Technology, Chinese Academy of Sciences during 2011-2013. Now he is a Ph.D. student in Harbin Institute of Technology Shenzhen Graduate School, and his research interest includes computer vision, computing intelligence, vehicle navigation.

Jeng-Shyang Pan received the B. S. degree in Electronic Engineering from the National Taiwan University of Science and Technology in 1986, the M. S. degree in Communication Engineering from the National Chiao Tung University, Taiwan in 1988, and the Ph.D. degree in Electrical Engineering from the University of Edinburgh, U.K. in 1996. Currently, he is a Dean for College of Information Science and Engineering, Fujian University of Technology and a director of Innovative Information Industry Research Center, Harbin Institute of Technology Shenzhen Graduate School, China. He jointed the editorial board of LNCS Transactions on Data Hiding and Multimedia Security, Journal of Computers, Journal of Information Hiding and Multimedia Signal Processing etc. His current research interests include soft computing, information security and signal processing.

Abdulhameed Alelaiwi is a vice dean for technical affairs, Scientific Research Deanship, King Saud University and a faculty member in Software Engineering Department, College of Computer and Information Sciences, KSU. He holds a Ph.D. in the field of Software Engineering from the department of software engineering, Florida Tech Univ., USA, 2002. He obtained his M.Sc. in the field of software engineering from Florida Tech University in 1998. He worked in the industry around 7 years, before joining King Saud University. He continues to consult with local corporations in the areas of Software Engineering, E- Government, and Information security. Dr. Alelaiwi has been consulting with many companies as well as government sites such as Ministry of Labor, Ministry of Defense, and Ministry of Information. His industry experience includes applications of the state-of-the-art techniques in solving software engineering, project engineering problems at Ministry of Labor, where he was Minister Consultant and the director of computer center at the same time. Before that, at Vinnell Corporation, Dr. Alelaiwi has developed complex algorithms and simulation programs in solving military related problems.