Neural Comput & Applic (2015) 26:1725–1738
DOI 10.1007/s00521-015-1836-9

ORIGINAL ARTICLE

# A heuristic optimization method inspired by wolf preying behavior

Simon Fong · Suash Deb · Xin-She Yang

Received: 16 September 2014 / Accepted: 23 January 2015 / Published online: 10 February 2015

**Abstract** Optimization problems can become intractable when the search space undergoes tremendous growth. Heuristic optimization methods have therefore been created that can search the very large spaces of candidate solutions. These methods, also called metaheuristics, are the general skeletons of algorithms that can be modified and extended to suit a wide range of optimization problems. Various researchers have invented a collection of metaheuristics inspired by the movements of animals and insects (e.g., firefly, cuckoos, bats and accelerated PSO) with the advantages of efficient computation and easy implementation. This paper studies a relatively new bio-inspired heuristic optimization algorithm called the Wolf Search Algorithm (WSA) that imitates the way wolves search for food and survive by avoiding their enemies. The WSA is tested quantitatively with different values of parameters and compared to other metaheuristic algorithms under a range of popular non-convex functions used as performance test problems for optimization algorithms, with superior results observed in most tests.

**Keywords** Metaheuristic · Bio-inspired optimization · Wolf Search Algorithm

## 1 Introduction

An optimization problem generally aims to find $x^{\text{opt}} = \max_{x \in X} f(x)$ where $\mathbb{R}^n$ is the search space and $f(x)$ is a fitness function measuring the goodness of the solution. The global optimum represents a best solution $x^{\text{opt}}$ that is assumed to exist in the problem space. In many real-life applications, the optimization functions may not behave well mathematically and hence do not always conceive a globally convex shape (see Fig. 1).

In such cases, especially when the data carry high-dimensional variables, the optimization problems can be complex and the problem sizes may thwart efficient calculation. For example, in the traveling salesman problem, the search space of candidate solutions grows more than exponentially as the size of the problem increases, which makes an exhaustive search for the optimal solution infeasible. A heuristic optimization method is a heuristic strategy for searching the search space of an ultimately global optimum in a more or less intelligent way [1]. This is also known as a metaheuristic or stochastic optimization. Metaheuristics are grounded in the belief that a stochastic, high-quality approximation of a global optimum obtained at the best effort will probably be more valuable than a deterministic, poor-quality local minimum provided by a classical method or no solution at all. Iteratively, it optimizes a problem by attempting to improve the candidate solution with respect to a given measure of quality defined by a fitness function. It first generates a candidate solution $x^{\text{candidate}}$, and as long as the stopping criteria are not met, it checks its neighbors against the current solution (SELECT

S. Fong (✉)
Department of Computer and Information Science, University of Macau, Taipa, Macau SAR
e-mail: ccfong@umac.mo

S. Deb
Department of Computer Science and Engineering, Cambridge Institute of Technology, Ranchi, India
e-mail: suashdeb@gmail.com

X.-S. Yang
School of Design engineering and Mathematics, Middlesex University, London, UK
e-mail: x.yang@mdx.ac.uk

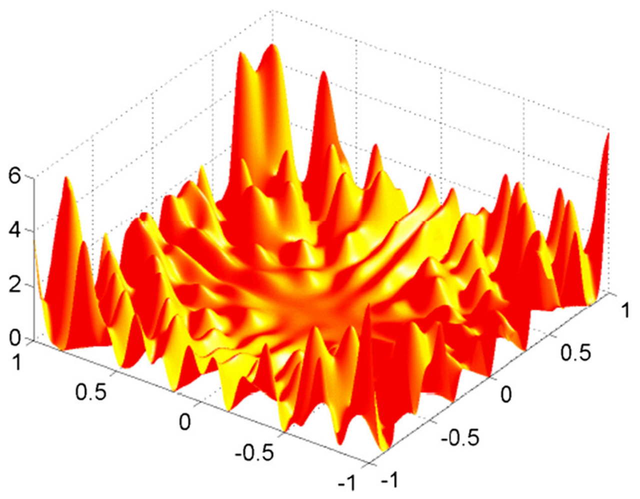

**Fig. 1** Example of a non-convex optimization function with complex local optima

$x^{\text{neighbor}} \in \mathbb{N}(x^{\text{candidate}})$). The candidate solution is updated with its neighbor if it is better (IF $f(x^{\text{neighbor}}) < f(x^{\text{candidate}})$ THEN $x^{\text{candidate}} = x^{\text{neighbor}}$), such that the global optimum at the end is $x^{\text{opt}} = x^{\text{candidate}}$. As such, metaheuristic algorithms are often based on local search methods in which the solution space is not explored systematically or exhaustively, but rather a particular heuristic is characterized by the manner in which the walk through the solution space is organized. Some computer science researchers have recently invented a collection of bio-inspired metaheuristic algorithms, including firefly [2], cuckoos [3], bats [4] and accelerated particle swarm optimization (PSO) [5]. These bio-inspired metaheuristic algorithms have local search methods that are largely based on the movement patterns of animals and insects found in nature. Their performance in heuristic optimizations has proven superior to that of many classical metaheuristic methods such as genetic algorithms (GAs), simulated annealing, ant colonies and tabu search.

This paper reports a relatively new bio-inspired metaheuristic algorithm, the Wolf Search Algorithm (WSA), which is based on wolf preying behavior. WSA is different from the aforementioned bio-inspired metaheuristics because it simultaneously possesses both individual local searching ability and autonomous flocking movement. In other words, each wolf in WSA hunts independently by remembering its own trait and only merges with its peer when the peer at sight is in a better position. In this way, long-range inter-communication among the wolves which are the searching agents for candidate solutions is eliminated because wolves are known to stalk their prey in silence. Assembly depends on visual range. Therefore, the swarming behavior of WSA, unlike most bio-inspired algorithms, is delegated to each individual wolf rather than to a single leader, as in PSO [4], fish [6] and firefly [2]. Effectively, WSA functions as if there are multiple leaders swarming from multiple directions to the best solution, rather than a single flock that searches for an optimum in one direction at a time. The appearance of a hunter that corresponds to each wolf is added at random. On meeting its hunter, each wolf jumps far out of its hunter's visual range to avoid being trapped in local optima by the algorithm's design.

This paper is organized into the following sections. Section 2 explains wolf preying behavior in layman's terms. Section 3 provides a detailed description of the algorithm's logics and operation. Section 4 reports on the experiment and discusses the results. Section 5 concludes the paper.

## 2 Background

Wolves are social predators that live in nuclear families consisting of a mated couple, their offspring and occasionally adopted immature wolves. Wolves typically commute in groups of 5–11, which is different from PSO and fish swarm, which usually move in relatively large groups. Wolves communicate over long distances by howling, but remain silent and use stealth when hunting prey together. Unlike ants in ant colony optimization, which use pheromones to communicate with their peers about food traits, WSA forgoes this kind of communication and collective stigmergy, which shortens the run time of the search.

Although wolf packs cooperate strategically when bringing down prey, they do not do so in the same way as lionesses because, unlike lions, wolves rarely remain with their pack for more than 2 years. This leaves them less time to learn how to hunt cooperatively. Wolves have therefore developed unique, semi-cooperative characteristics; that is, they move in a group in a loosely coupled formation, but tend to take down prey individually. This detail is important because some optimization algorithms, such as those that are swarm-based, focus on group coordination, whereas algorithms that emphasize individual movements fall on the other end of the spectrum. As a synonym in computing, WSA naturally balances scouting the problem space in random groups (breadth) and searching for the solution individually (depth).

When hunting, wolves will attempt to conceal themselves as they approach their prey. This characteristic prompts the searching agents in WSA to always look for and move to a better position in the same way that wolves continuously change their positions for better ones with more shelter, fewer terrain obstacles or less vulnerability. When hunting, wolves simultaneously search for prey and watch out for threats such as human hunters or tigers. Each wolf in the pack chooses its own position, continuously moving to a better spot and watching for potential threats. WSA is equipped with a threat probability that simulates incidents of wolves bumping into their enemies. When this happens, the wolf dashes a great distance away from its current position, which helps break the deadlock of getting stuck in local optima. The direction and distance they travel when moving away from a threat are random, which is similar to mutation and crossover in GA when changing current solutions while evolving into a better generation.

Another important feature of wolf hunting is that they hunt by wearing down their prey with short chases. When chasing small prey, wolves attempt to catch up with their prey as soon as possible, whereas in the pursuit of larger animals, the chase is prolonged to exhaust the prey. Likewise, WSA is able to track moving or evolving global optima when the best position is non-stationary, whereas convergence is quicker when the global optima are not of a significantly greater value than the best of the local optima. Otherwise, the convergence slows down to widen the breadth of the search when the final optimum is expected to be of great value. This speed of convergence is somewhat similar to that of a hill-climbing algorithm.

Wolves have an excellent sense of smell and often locate prey by scent. Similarly, each wolf in the WSA has a sensing distance that creates a sensing radius or coverage area—generally referred to as visual distance. This visual distance is applied to the search for food (the global optimum), an awareness of their peers (in the hope of moving into a better position) and signs that enemies might be nearby (for jumping out of visual range). Once they sense that prey is near, they approach quickly, quietly and very cautiously because they do not wish to reveal their presence. In search mode, when none of the abovementioned items are detected within visual range, the wolves move in Brownian motion (BM), which mimics the random drifting of particles suspended in fluid.

## 3 Formulation of the Wolf Search Algorithm

### 3.1 Logics

Based on wolves' hunting behavior, as described above, we present the three rules that govern the logics of the WSA.

1. Each wolf has a fixed visual area with a radius defined by $v$ for $X$ as a set of continuous possible solutions. In 2D, the coverage would simply be the area of a circle by the radius $v$. In hyperplane, where multiple attributes dominate, the distance would be estimated by Minkowski distance, such that $v \leq d(x_i, x_c) = \left(\sum_{k=1}^{n} \left|x_{i,k} - x_{c,k}\right|^{\lambda}\right)^{1/\lambda}, x_c \in X$ where $x_i$ is the current position; $x_c$ are all the potential neighboring positions near $x_i$ and the absolute distance between the two positions must be equal to or less than $v$; and $\lambda$ is the order of the hyperspace. For discrete solutions, an enumerated list of the neighboring positions would be approximated. Each wolf can only sense companions who appear within its visual circle and the step distance by which the wolf moves at a time is usually smaller than its visual distance.
2. The result or the fitness of the objective function represents the quality of the wolf's current position. The wolf always tries to move to better terrain. But rather than choosing the best terrain, it opts to move to better terrain that already houses a companion. If there is more than one better position occupied by its peers, the wolf will choose the best terrain inhabited by another wolf from the given options. Otherwise, the wolf will continue to roam randomly in BM.
3. At some point, it is possible that the wolf will sense an enemy. The wolf will then escape to a random position far from the threat and beyond its visual range.

Given these rules, we summarize the WSA in pseudocode:

There are some conceptual similarities between WSA and the artificial fish school algorithm [6]. Both use the visual distance concept, and both types of searching agents only care about the things that are happening within their visual circle. However, in the artificial fish school algorithm, the fish are always following and swarming as a whole when searching because the algorithm does not contain the WSA's individual search capabilities or jump out mechanism.

The function `Generate_new_location()` returns a valid position that is unseen by a wolf, given its short-term memory. WSA implements one step as the previous step, which means that the new location generator randomly produces a new position from a pool of candidates within $s$ step from the wolf's current position with the exception of the position that the wolf occupied in the previous cycle. This generation of each candidate's pool of positions follows the rules of BM. In addition, the length of the back-track trail can be increased to a greater number assuming that the wolf has more memory about its walked path. This can be compromised at significant computational cost and time when the wolf population is substantial.

### 3.2 Following and approaching other wolves

WSA assumes that the result/fitness of the objective function reflects the quality of a terrain position that will eventually lead to food. This quality can be defined as either secludicity from predators, higher ground from which it is easier to hunt, or another similar benefit. The intention behind a wolf's decision to change location is to simultaneously secure an increased chance of finding food and a decreased chance of being hunted. Wolves are expected to trust other wolves, because they never prey on each other; therefore, a wolf will only move into terrain inhabited by another wolf when that terrain is better. If the new position is better, the incentive is stronger provided that it is already inhabited by a companion wolf. There is another factor that must be considered, specifically the distance between the current wolf's location and its companion's location. The greater this distance, the less attractive the new location becomes, despite the fact that it might be better. This decrease in the wolf's willingness to move obeys the inverse-square law. Therefore, we get a basic formula of betterment $r = \frac{I_o}{r^2}$, where $I_o$ is the origin of food (the ultimate incentive) and $r$ is the distance between the food or the new terrain and the wolf. There is a similar formula for calculating attractiveness in the firefly algorithm [4]. It is also added with the absorption coefficient, such that using the Gaussian equation, the formula is $\beta(r) = \beta_o e^{-r^2}$ where $\beta_o$ equals $I_o$. For simplicity, we use the following as the incentive formula in our wolf search:

$$\beta(r) = \beta_o e^{-r^2} \tag{1}$$

Given that all wolves want to move to better positions inhabited by their peers and based on the assumption that their visual distance is good but limited, each wolf can only spot its peers when they enter the initial wolf's sensing coverage. The wolf cannot sense and therefore will not move toward companions beyond this range. Furthermore, if the positions of a wolf's peers are no better than its current position, then there is no incentive for the wolf to move. Other wolves will, however, eventually merge to the wolf's current position if it is the best option. The tri-step operation works in the following way. First, each wolf checks the quality of its companions' positions within its visual range and the best location out of all qualified locations is identified. Second, the wolf compares its own position with that best location. Third, the wolf moves to that best location and will stay with its companion if there is a gain in adopting that position. Otherwise, the wolf continues preying via a BM random walk with an incremental step size that is shorter than its visual distance. The preying operation will be introduced in detail in the next section. Afterward, the wolf moves to join its companion and the movement is implemented using the following formula:

$$x(i) = x(i) + \beta_o e^{-r^2}(x(j) - x(i)) + escape() \tag{2}$$

where *escape*() is a function that calculates a random position to jump to with a constraint of minimum length; $x$ is the wolf, which represents a candidate solution; and $x(j)$ is the peer with a better position as represented by the value of the fitness function. The second term of the above equation represents the change in value or gain achieved by progressing to the new position. $r$ is the distance between the wolf and its peer with the better location. Step size must be less than the visual distance.

## 3.3 Preying

Wolves pursue their prey using a stalking process. In real life, a wolf scouts every part of its territory to search for ungulates and its movement pattern obeys BM because wolves who live and hunt together remain as a nuclear family rather than as a big colony. No two wolves come close to each other in random movement unless they are feeding on the same food or using the same shelter (e.g., holed up in a cave). WSA features three different types of preying behavior that take place in sequence. In the context of an algorithm, these three types of behavior occur in an atomic manner in each iteration or generation of execution.

1. Preying initiatively: The wolf feeds on food, which represents the ultimate objective of the optimization function. This step essentially allows the wolf to check its visual perimeter to detect prey. The step is placed at the beginning of the execution loop, and it repeats after checking whether the current location should be changed or after a random step in the random walk, such that the wolf will constantly be looking for its prey. Once the prey is spotted within the wolf's visual distance, it will diligently move step by step toward the prey (the food) that has the highest fitness, in which circumstance the wolf will omit looking out for its companions. In WSA, this is reflected by the fact that the wolf will change its own position for that of the prey, which has the highest value, and because no other position is higher than the highest, the wolf will maintain this direction.
2. Prey passively: If the wolf does not find any food or better shelter inhabited by a peer in the previous step, then it will prey passively. In this passive mode, the wolf only stays alert for incoming threats and attempts to improve its current position by comparing it to those of its peers.
3. Escape: Wolves have numerous enemies in nature. When a threat is detected, the wolf escapes very quickly by relocating itself to a new position with an escape distance that is greater than its visual range. The emergence of threats is modeled randomly at a probability defined at will by the user. Escape is an important step that helps keep all of the wolves from falling into and getting stuck at a local optimum. We use mathematics to define the abovementioned types of preying behavior:

```
Objective function $f(x)$, $x=(x_1,x_2,..x_d)^T$
Initialize the population of wolves, $x_i(i=1,2,..,W)$
Define and initialize parameters:
$r$ = radius of the visual range
$s$ = step size by which a wolf moves at a time
$\alpha$ = velocity factor of wolf
$p_a$ = a user-defined threshold [0..1], determines how frequently an enemy appears

WHILE ($t$<generations && stopping criteria not met)
  FOR $i$=1:$W$  // for each wolf
    Prey_new_food_initiatively();
    Generate_new_location();
    // check whether the next location suggested by the random number generator is new. If not, repeat
       generating random location.
    IF($dist(x_i,x_j)<r$&&$x_j$ is better as $f(x_i)<f(x_j)$)
        $x_i$ moves towards $x_j$// $x_j$ is a better than $x_i$
    ELSE IF
        $x_i$ = Prey_new_food_passively();
    END IF
    Generate_new_location();
    IF($rand()>p_a$)
        $x_i = x_i + rand() + v$; // escape to a new pos.
    END IF
  END FOR
END WHILE
```

**Fig. 2** Pseudocode for the WSA algorithm

$$\text{if moving} = \begin{cases} x(i) = x(i) + \alpha \cdot r \cdot rand() & Prey \\ x(i) = x(i) + \alpha \cdot s \cdot escape() & Escape \end{cases} \quad (3)$$

where $x(i)$ is the wolf's location; $\alpha$ is the velocity; $v$ is the visual distance; $rand()$ is a random function whose mean value distributed in $[-1,1]$, $s$ is the step size, which must be smaller than v; and $escape()$ is a custom function that randomly generates a position greater than $v$ and less than half of the solution boundary. Preying initiatively and preying passively both use the upper part of the formula for movement, whereas escape uses the lower part. Additional modifications, such as ensuring that the walk is in BM and considering the merging of peer positions, are also implemented according to the logics defined in Fig. 2.

Figure 3 illustrates the wolves' movement. $x(i)$ is in range with $x(j)$ and $x(m)$. $x(i)$, however, is moving toward $x(j)$ because it has a better terrain or is closer to the food (as represented by the better fitness function value). $x(m)$ will follow $x(i)$, and $x(k)$ will leap a large distance away when the threat alert is trigged by the probability. Otherwise, it is probable that $x(k)$ may sense $x(j)$ in its range and merge if $f(x(j)) > f(x(k))$. $x(j)$ will ignore its nearby peers and move toward the food, which has the highest fitness value. $x(l)$ is roaming in BM some distance away.

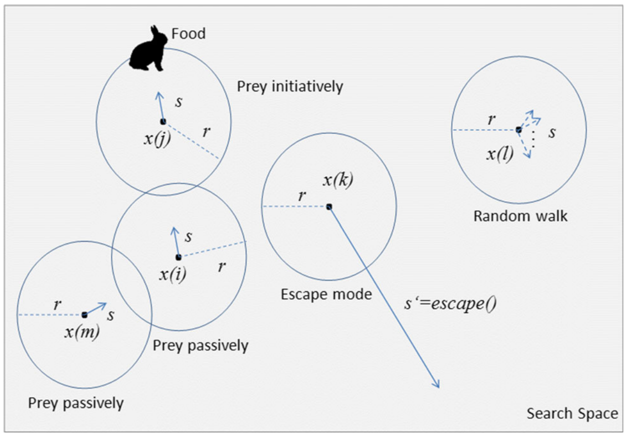

**Fig. 3** Snapshot of WSA in action

## 4 Validation and comparison

### 4.1 Experiments for optimization algorithm validation

To validate the efficacy of this new algorithm, we implemented WSA in MATLAB and tested various famous optimization functions. They have been adopted popularly in the optimization research community for performance validation. In general, the functions can produce a large

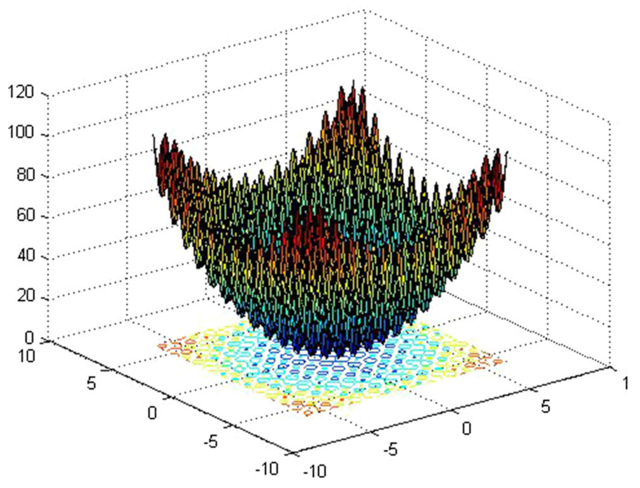

**Fig. 4** Overview of Rastrigin's function

multidimensional search space where both local minima and global minimum are mathematically distributed. The fitness functions of the testing functions are shown in the "Appendix." Here, we use one of them to demonstrate WSA converges. Rastrigin's function [7] is a non-convex function for testing an optimization algorithm. There are many local minima in Rastrigin's function, which makes it highly multimodal. The function is defined as follows:

$$f(x) = 10n + \sum_{i=1}^{n} \left[x_i^2 - 10\cos(2\pi \cdot x_i)\right] \tag{4}$$

where $x_i \in [-5.12, 5.12]$. The visualization of Rastrigin's function in 3D is shown in Fig. 4. The global minimum of Rastrigin's function is $f(\cdot) = 0$ at (0, 0). We set the parameters $W = 20$ wolves, step size = 1, visual = 2, escape probability = 0.25 and a tolerance criterion equal to $\pm 1.0e^{-8}$, under which circumstances WSA eventually reaches the minimum. Given $\varepsilon < 10^{-8}$, the optimization functions are executed iteratively until the difference in best fitness between the current and the previous iteration is smaller than $10^{-8}$. The initial step and final result are shown graphically in Fig. 5a, b, respectively. The blue points are the locations of the wolves. The computing environment is a MacBook Pro (with CPU: 2.3GHZ, RAM: 4 GB).

Although the Rastrigin's test function has numerous local minima distributed all over the space, after running the algorithm, almost all of the wolves clearly gather at (0, 0), which suggests that WSA deals with multimodal problems pretty well given the right set of parameters; that is, they are able to find the global optimum and converge eventually there. The same convergence is observed in the other testing functions.

The experiment is then extended to test the effects of its parameters with varying values, such as population,

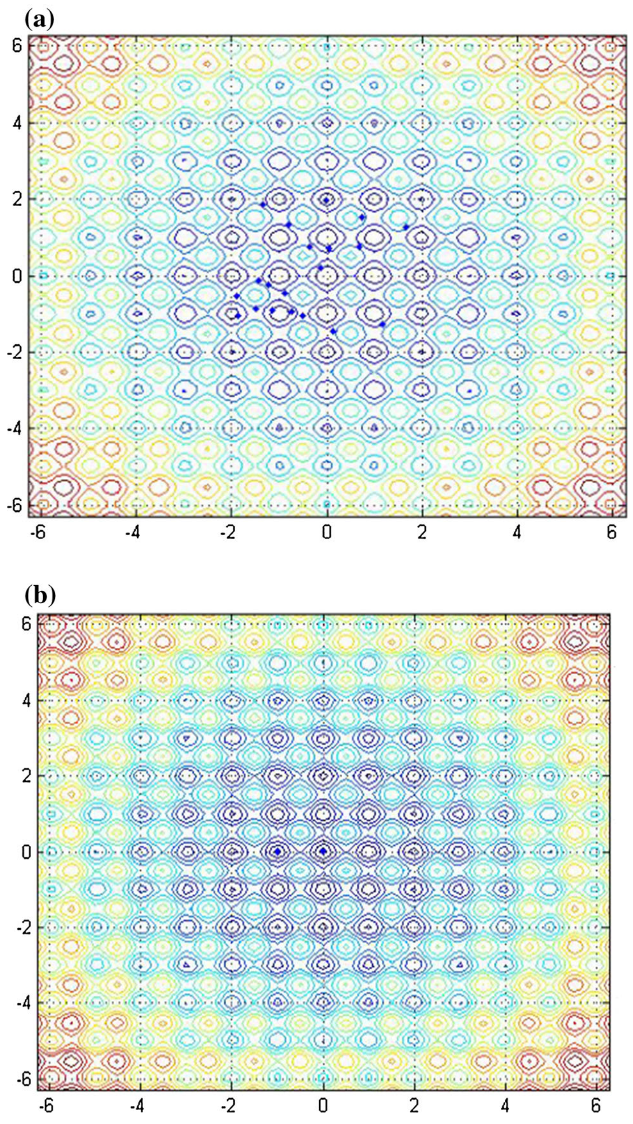

**Fig. 5** **a** Wolves' locations at initial step. **b** The wolves' locations at final step

memory tenure, escape probability and visual distance. Six testing functions from "Appendix" are used. In common, these testing functions have the global minima: $x^* = (0,\ldots,0)$, $f(x^*) = 0$. The six functions are visualized in three dimensions as shown in Fig. 6.

The first parameter to be tested is population which accounts for the total number of wolves as search agents to be deployed in the search. Intuitively, the more search agents in use, the greater the computational cost the algorithm incurs. With a large number of distributed agents searching over the space, a better final fitness value should be obtained. Although the search agents could be potentially programmed and operating in parallel, the implementation in this experiment here is sequential execution. The wolf agents take turn to compute their own fitness functions and

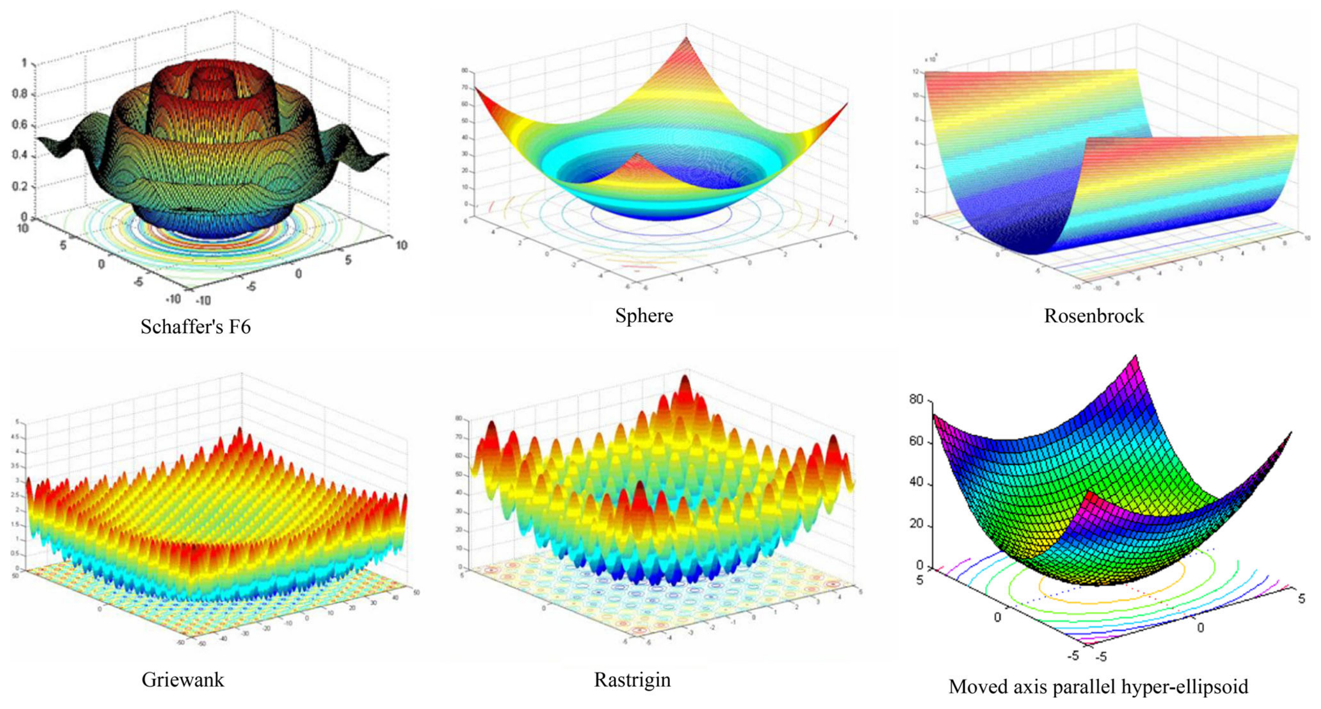

**Fig. 6** Visualization of the six testing functions

update their moves one iteration at a time, until the stopping criteria are met. The performance indicators hence are the fitness and the time consumption. Given the other parameters at default values as above, the relation between the fitness and time would be investigated across the six testing functions, in the hope of finding a balance. Six diagrams of various populations that show the fitness–time relations are in Fig. 7a–f. The unit of time is in second, which is the total runtime elapsed from the initial to the final iterations. The fitness values are graphed in logarithmic scale for easy visual comparison. The wolf agents converge near the global optima for all the cases. The resultant fitness is the best fitness retained from the last iteration before convergence. Unlike deterministic functions that find an absolute global minimum by brute force but it may take forever, the WSA that is stochastic in nature locates an approximate solution very close to the absolute best.

The charts from Fig. 7a–f indeed show a general relation between fitness and time as the population increases—better fitness is gained at the cost of longer runtime. The time cost is about the same for all the testing functions. At the population of 100 wolves, the runtime ranges from 1,209 to 1,814 s. However, the curves of the fitness values differ in each testing functions with different gradients. A steep fitness curve implies that the marginal gain in terms of fitness by using more search agents is higher. Testing functions that produce steep curves are Rosenbrock, Griewank and Rastrigin. Coincidentally, these functions in common have relatively steeper parabolic shapes and filled more with challenging local minima and maxima. Some fitness curve takes the form of steps, like the one tested in Schaffer's function. The large steps in this curve resemble the folding shape of the Schaffer's function (c.f. Fig. 6). Except Schaffer's and moved axis which have relatively apparent hill-climbing and flat plateaus, the fitness curves reach the best optima only in high populations. The optimal population should be one that is located at the intersection of the fitness curve and the time consumption trend, maintaining a balance between solution quality and time requirement. For example, Schaffer's and moved axis have an optimal population at $W = 35$. The results from this population experiment hint that for multimodal functions (and problems), having a large population is definitely advantageous in yielding better solutions, as evidenced by the gradient of the curves. A relatively small population would be sufficient for easy problems of flat plateau, for a quality solution considering the trade-off of time cost.

The experiment now proceeds to evaluate the effects of intensification and diversification. Intensification relates to the local search conducted by the search agents. It is about how smart the agents scout for better solutions around and near their current positions. Wolves in nature can make use of stigmata such as their pawprints that marked their previous trails, for improving their search direction. The pawprints of the wolves which are analogous to past trails are maintained in a global memory in our implementation (with the MP, minus previous parameter setting), with the visited places in the search place and their associated

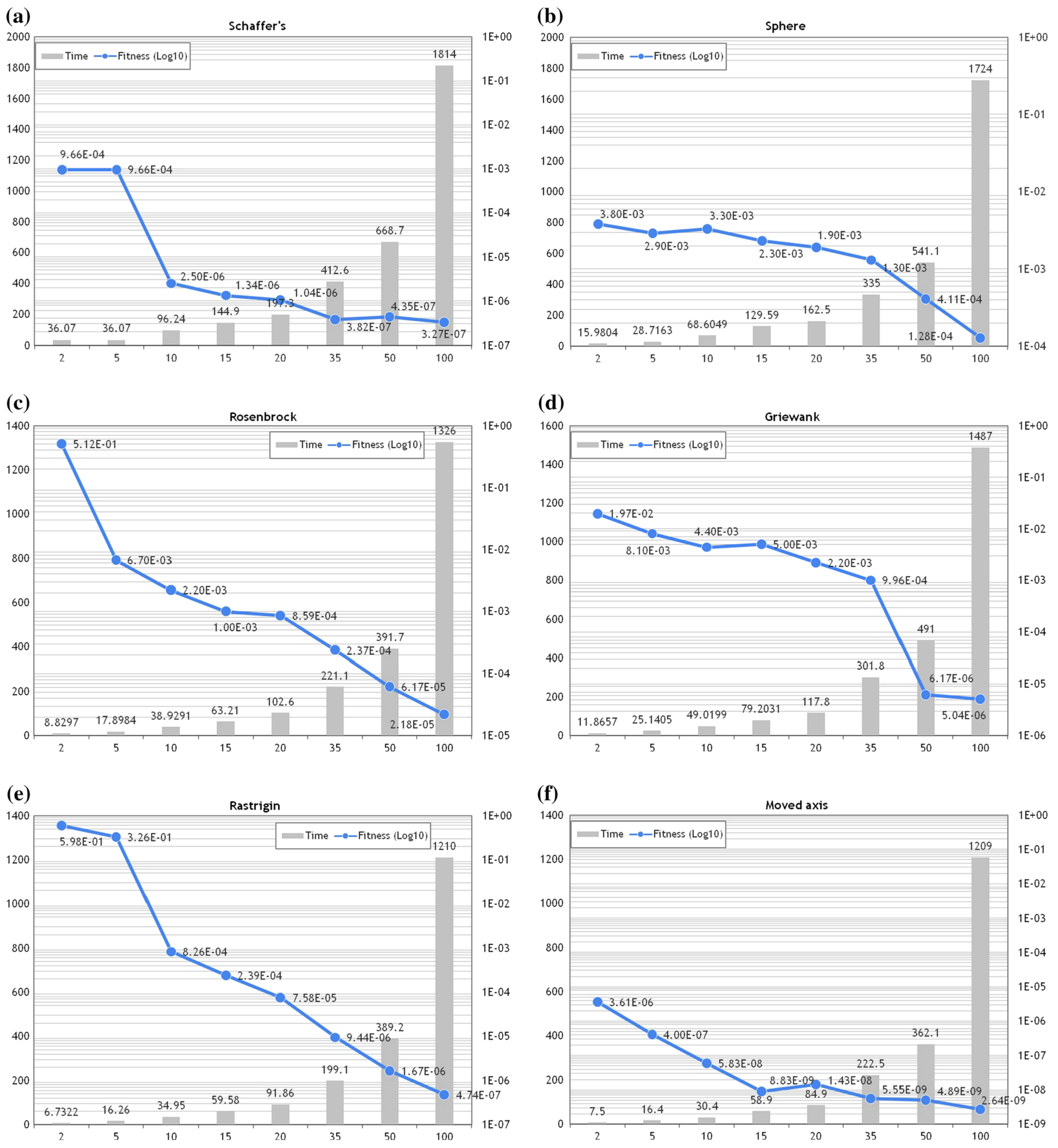

**Fig. 7** **a** Effect of population by function Schaffer's F6. **b** Effect of population by function sphere. **c** Effect of population by function Rosenbrock. D Effect of population by function Griewank. **e** Effect of population by Rastrigin. **f** Effect of population by function moved axis

fitness values in store. Instead of memorizing the best position and the highest fitness value in every iteration, a tabu list is used to accumulate all the visited places except the best ones from previous iterations. In other words, the relatively less desirable places which have already been visited by the wolves are collectively memorized from iteration to iteration. So the search agents will not waste time and computational resource in going back there again from their random walk. By not considering those tabu places, the search agents are likely to move away from undesirable places (out of the local search) and swarm into a better terrain in general. The MP parameter is in use in

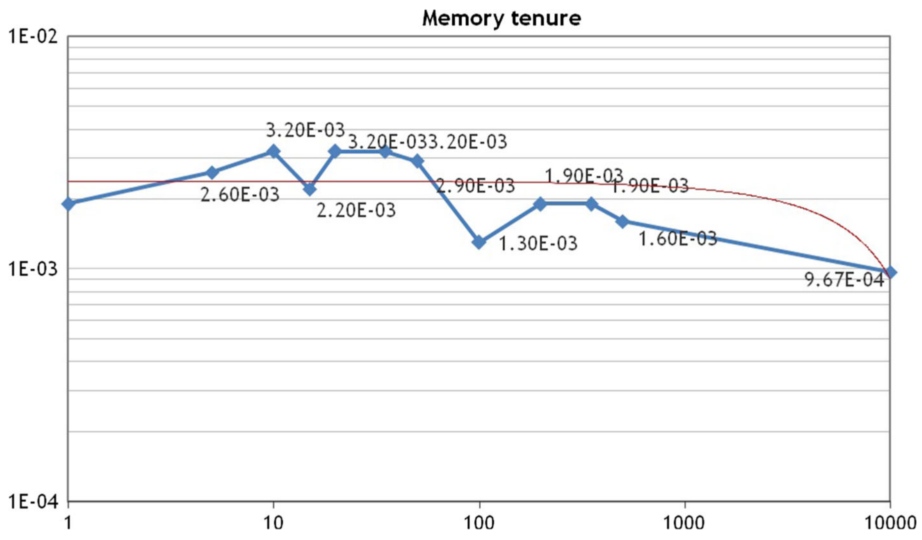

**Fig. 8** Effect of memory tenure over the testing functions

this experiment. The MP value is equivalent to the memory tenure (long- or short-term memory) for the wolves remember how many previous steps they took in the search space especially the undesirable ones (with low fitness values). The overall fitness is compared with the length of the memory tenure (which is proportional to MP) or the memory strength, as shown in Fig. 8. The parameter setting is the same as those reported earlier in this paper, and the testing function used is Schaffer's F6. Various memory tenures are used for the wolves to compute a final best possible fitness value in each run. The fitness curve is plotted at logarithmic scale. As it can be seen in Fig. 8, along with some fluctuations possibly due to the probabilistic nature of the search operation, the fitness indeed improves with increasing memory tenure. That means the stronger the memory that the wolves possess, the better solution they can produce. To illustrate this effect more clearly, a linear regression trend in red color is added to the chart. The trend dips, however, only after certain length of memory tenure (i.e., MP $> \approx 5{,}000$). This shows that the memory effect has little impact on the fitness if the memory tenure is relatively short. A considerable amount of memory strength is required for the fitness improvement to emerge. This memory-based version of WSA certainly yields better fitness values; however, the cost of heavy memory may need to be taken into consideration in implementation.

The parameter of escape probability controls the extent of diversification over the wolves' movements. The higher the escape probability means more often the wolves will transport to other dimensions afar. The experiment again is conducted using the default parameters as earlier, except the escape probability is varied. The fitness performance fluctuates considerably as the escape probability increases. The best fitness is attained when the escape probability is moderate (at $\approx 0.25$). This clue testifies that a moderate frequency of escape is good, for the wolves to balance between intensive local search and exploring new solutions afar. On the end where the escape probability is high, the wolves may not converge easily or not at all, leading to a forced termination when the maximum allowed number of iterations is reached. As a result, the wolves stopped at the relatively poor-quality solutions. To see the effect more clearly, a trend line is added. Too high the escape probability may not be desirable in optimization.

With respect to flocking or swarming characteristic, WSA relies on its visual range. This is another main factor that governs the swarming (merging) behavior of the wolves. The effect of this important parameter visual distance is evaluated, and the results are shown in Fig. 10a–f. It can be generally noticed that similar to the effect of escape probability in Fig. 9, too large the visual range does not yield good fitness. Having too large, the visual distance means the wolves get merged together too easily; vice versa too short the visual range the wolves seldom get to merge. The optimal visual distance is usually found at the moderate range of visual distance. An interesting phenomenon is observed that the fitness representing the quality of the found solution declines by different extents in different testing functions. Moved axis yields poor solution as soon as the visual distance grows, probably due to its smooth parabola search space—the wolves get merged too easily if they are not shortsighted. Similar trends are observed from Schaffer's, Sphere and Rosenbrock functions, where the fitness worsens by rising up the fitness curve gradually in log scale as the visual distance increases. In particular, Sphere that is a very smooth parabolic function has a smooth and consistent curve showing the decline of the solution quality gradually. However, in contrast, in the highly multimodal functions

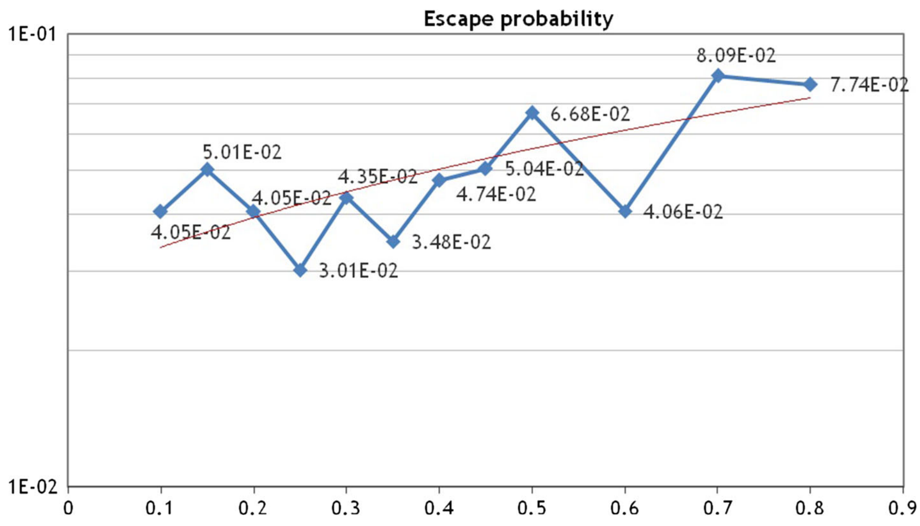

**Fig. 9** Effect of escape probability over the testing functions

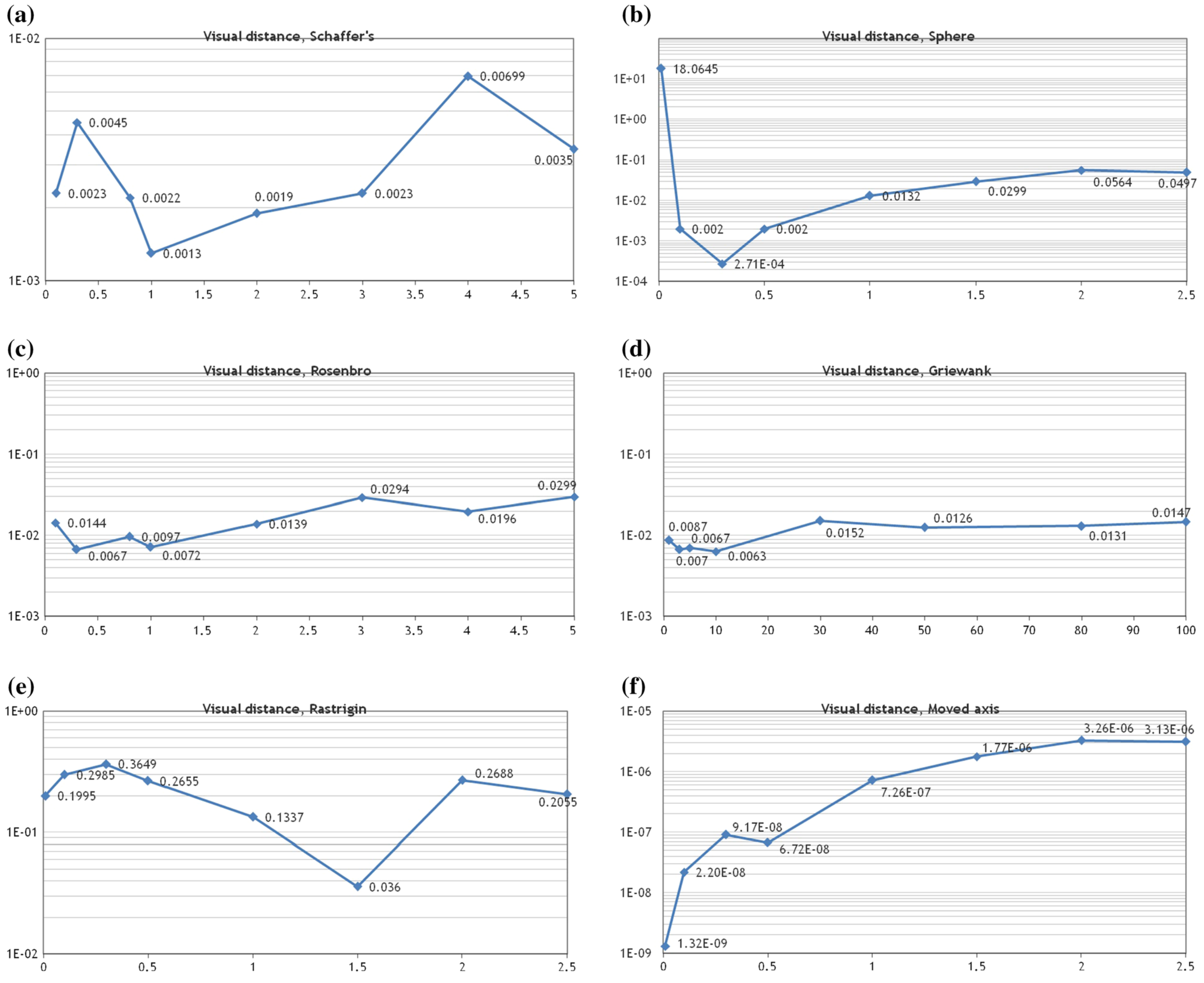

**Fig. 10** **a** Effect of visual distance by function Schaffer's F6. **b** Effect of visual distance by function Sphere. **c** Effect of visual distance by function Rosenbrock. **d** Effect of visual distance by function Griewank. **e** Effect of visual distance by function Rastrigin. **f** Effect of visual distance by function moved axis

**Table 1** Comparison using fixed tolerance threshold

| Function/algorithm | PSO | WSA | WSA-MP |
|---|---|---|---|
| Griewangk's | 1,070 ± 633 (100 %) | 398 ± 390 (100 %) | 365 ± 303 (100 %)* |
| Sphere model | 1,670 ± 1,024 (100 %) | 549 ± 542 (100 %)* | 600 ± 623 (100 %) |
| Rastrigrin's | 24,206 ± 16,000 (100 %) | 14,813 ± 2,170 (100 %)* | 15,579 ± 16,466 (100 %) |
| Schaffer's F6 | 34,046 ± 7,461 (97 %) | 6,537 ± 4,671 (100 %) | 3,306 ± 4,867 (100 %)* |
| Moved axis parallel | 4,779 ± 3,186 (100 %) | 2,180 ± 5,764 (100 %)* | 4,518 ± 6,691 (100 %) |
| Bohachevsky's | 3,993 ± 3,186 (100 %)* | 4,019 ± 4,001 (100 %) | 5,546 ± 5,236 (100 %) |
| Michalewicz's | 14,283 ± 8,615 (92 %) | 4,295 ± 3,357 (100 %) | 4,001 ± 3,342 (100 %)* |
| Rosenbrock's | 15,197 ± (9,554) (100 %) | 6,003 ± 7,018 (100 %) | 5,197 ± 5,740 (100 %)* |

**Table 2** Comparison using 1,000 fixed evaluations

| Function/algorithm | GA | PSO | WSA | WSA-MP |
|---|---|---|---|---|
| Griewangk's | 0.1,860 | 2.7559e−8 | 1.9126e−7 | 1.0690e−7 |
| Sphere model | 0.0019 | 9.5715e−8 | 2.8765e−8* | 1.6620e−7 |
| Rastrigrin's | 0.0677 | 1.5829e−5 | 1.0297e−5 | 1.2153e−5 |
| Schaffer's F6 | 0.0090 | 3.8226e−4 | 1.930e−4 | 9.7579e−5* |
| Moved axis parallel | 0.0078 | 5.9182e−7 | 1.8917e−6 | 1.8770e−6 |
| Bohachevsky's | 0.0081 | −0.2400* | −0.2400* | −0.2400* |
| Michalewicz's | 0.0004 | −1.1801* | −1.8012* | −1.8012* |
| Rosenbrock's | 0.3060 | 8.2129e−6 | 2.5192e−6 | 6.1665e−7 |

Griewank and Rastrigin, the fitness curves versus the increase in visual distance perceived to be a flat line when averaging out a few sharp drops and rises. It looks as if the length of visual distance has relatively little effect on the fitness. From Fig. 6, it can be seen that these two particular testing functions are mathematically toughest that have complex and dense clusters of local optima. These complex characteristics contribute to the confusion of the wolves' search regardless of the visual distances. As a concluding remark, a moderate range of visual distance helps achieving a good fitness hence quality solution.

## 4.2 Comparison of WSA to PSO and GA

PSO [4] and GA [8] are two classical nature-inspired optimization algorithms to which we compared WSA. Numerous studies have shown that PSO outperforms GA, and the GA and PSO used in this comparison are standard versions with default parameters used for benchmarking purposes.

Our following experiment uses two approaches to test WSA. The first approach uses a fixed tolerance criterion, $\varepsilon < 10^{-5}$. The optimization functions are executed iteratively until the difference in best fitness between the current and the previous iteration is smaller than $10^{-5}$. The first approach only tests PSO and WSA because GA falls too easily into the local minimum, and the tolerance is too small. In the second approach, we run for a fixed number of iterations of 1,000 rounds, regardless of yielded improvements, and then compare the mean of premature fitness after stop. Each algorithm is repeated 100 times to produce a meaningful statistical analysis. In addition to the six testing functions, Bohachevsky's and Michalewicz's functions are used too for they test the convergence at other points that represent their global minima than (0,…0).

In this experiment, the memory-based version of WSA, WSA with Step Minus Previous (WSA-MP) is implemented. In a scenario where a wolf takes a subsequent random step to escape from a threat or roam randomly, the wolf is likely to take any random step in any random direction except its old position from the previous round of calculations. This indicates that wolves possess a photographic memory that remembers not to backtrack. This variation in metaheuristics can enhance the search by avoiding a return to old states.

Table 1 shows the mean of the number of function evaluations with the ±sign as the standard deviation. In our implementation, the population of all three algorithms is 20 and the acceleration factor equals 1.5. For GA, the mutation probability of $p = 0.05$ and the crossover probability of 0.8, such that there is no elitism in our GA.

The above two tables indicate that WSA is much better than PSO, which is much better than GA. Table 1 shows that WSA uses less rounds of evaluations than PSO and can find the global optimum efficiently and accurately. Table 2 demonstrates that GA easily moves into a local minimum that PSO and WSA avoid. Moreover, WSA achieves more accurate results. The superior results are marked with a asterisk for easy observation in each entry of comparison.

## 5 Discussion of the characteristics of WSA

The WSA's features are discussed here with respect to its consideration as a metaheuristic algorithm candidate based on its main components as documented and tested in this paper. Four basic metaheuristic characteristics are used as references in the subsequent discussion, and WSA is compared to other metaheuristic algorithms based on these characteristics.

### 5.1 Trajectory method

Unlike most bio-inspired algorithms, the solutions tested (wolves) in WSA move in a 2-step or hybrid mode involving individual scans of immediately neighboring areas for food and better positions for peer merging. Trajectory is therefore distributed, and the directions are independent for each wolf. When neither food nor peer wolves are within visual distance, the walk is in Brownian movement.

### 5.2 Discontinuous method

In simulated annealing [9], threshold accepting [10] and tabu search algorithms [11], the full solution space is available for a new solution. Discontinuity induced by the generation of starting solutions, such as the genetic and ant colonies algorithms [12], corresponds to jumps in search space. WSA also has jumps in search space.

### 5.3 Population-based method

Similar to the genetic and ant colonies algorithms, but different from simulated annealing and tabu search algorithms, the population of searching agents all contribute to the collective experience. In WSA, the searching agents (wolves) merge at a solution when they cannot find anything better. Multiple wolves gathering at the same solution implies common agreement that the solution is good, which could technically be the global optimum or a very good local optimum. The random generator for each wolf, however, will probably relinquish it from the point to look for better solutions. A 'jumped out' wolf will eventually merge back into the crowd when no better situation is found. If the separate wolf does find a better solution, it will stay there and its peers will eventually shift to that solution as their random enemy hits and as long as that position is found to be the best.

### 5.4 Guided search (with memory use) or unguided search (memory-less) method

In a guided search, additional rules and hints are incorporated about where to search. For instance, the GA population represents the memory of recent search experience; the pheromone matrix in the ant colonies algorithm represents the adaptive memory of previously visited solutions; and the tabu list provides short-term memory. WSA lies between these search methods in achieving the minimum use of memory. In nature, a moving wolf occasionally looks back and remembers its past trail to a certain extent. In WSA, only the previous step (solution) is remembered by a searching agent, so the new solution will always be different from the previous one. The memory of the past trail can be extended from one step to as long as the memory constraint can hold. Conceptually, remembering the past trail to a certain length should provide better performance at the cost of memory and time. In addition, WSA uses no amount of information for intercommunications. Ants in ACO leave chemical marks (pheromones) for orientation and attracting the crowd and prefer trails with high pheromone concentrations that supposedly lead to better food. The wolves in WSA keep silent and depend largely on their sensing ability and instincts to decide whether their current positions are less desirable than those of their peers; if so, they give up their current positions and join their peers. Their positions and the relative distances to the food are then progressively updated, purely by heuristics.

Possible extensions, in addition to visual range, include that fact that each wolf can sense its peers using body heat and/or scent. Similar to the pheromones used in the ACO algorithm, WSA can flock more smoothly by having some wolves follow others. The other extension is that the step size should be adaptive rather than constant. Specifically, the proximity of food would accelerate convergence. The last extension suggests that when the wolves are in a bigger pack, the probability of an enemy emerging would decrease. In nature, this would translate as a pack of wolves being less afraid of an enemy (presumably, the enemy is an animal of similar capacity and size, not a human hunter holding a gun). Decreasing the chances that the searching agents will swoop out of a group that may have reached an optimum should speed up runtime. Nevertheless, there should be a balance between seeking better optimums and a quick convergence at a quality solution.

## 6 Conclusion and future work

Heuristic optimization methods have an edge over their classical counterparts because they can incrementally induce a globally optimum solution by using heuristics to efficiently search a large space. A special kind of heuristic optimization known as nature-inspired optimization or metaheuristics is gaining substantial popularity in the research community due to its advantages, which are applicable in computational intelligence, data mining and their applications. Borrowed from the wonders of nature, such algorithms computationally optimize complex search problems with superior performance and search efficiency compared to earlier optimization techniques. This paper presents a new metaheuristic algorithm, the WSA, which imitates the preying behavior of wolves and has displayed unique advantages in efficiency because each searching agent simultaneously performs autonomous solution searching and merging. Local optima are overcome when the searching agents leap far away upon being triggered by the random emergence of an enemy. In this paper, the performance in terms of fitness which is equivalent to the solution quality is thoroughly validated over different algorithmic parameters. Furthermore, WSA is tested against classical algorithms such as GA and PSO and it outperformed them in most of the testing cases. The WSA's potential contributions to finding optimal solutions in applications include but are not limited to the following: traveling salesman problems, quadratic assignment problems, job scheduling problems and sequential ordering problems. Readers who are interested in testing WSA with a wide spectrum of optimization problems are encouraged to download the WSA program written in Matlab for free, at www.researchgate.net/profile/Simon_Fong.

Future work should involve testing WSA with a non-stationary global optimum; that is, the point of the global optimum moves in time. This is common in moving data streams in which the concept drifts and therefore the position of the globally optimal point may also drift. WSA should be able to adapt to a moving global optimum, at least in principle, by choosing appropriate speed, visual range and step size parameters.

**Acknowledgments** The authors are thankful for the financial support from the research grant "Adaptive OVFDT with Incremental Pruning and ROC Corrective Learning for Data Stream Mining," Grant no. MYRG073(Y3-L2)-FST12-FCC, offered by the University of Macau, FST and RDAO.

## Appendix: fitness functions

We used the following test functions to test our algorithms.

1. Griewangk's function: In this test function, there are a lot of local minima,

$$f(x) = \sum_{i=1}^{n} \frac{x_i^2}{4000} - \prod_{i=1}^{n} \cos\left(\frac{x_i}{\sqrt{i}}\right) + 1$$

The global minimum is $f_{\min} = 0$ at (0,…,0), where the $-600 < x_i < 600$.

2. Sphere function:

$$f(x) = \sum_{i=1}^{d} x_i^2, \quad \text{where the } -5.12 < x_i < 5.12$$

The global minimum is $f_{\min} = 0$ at (0,…,0).

3. Rastrigin's function: This function is difficult due to its large search space and large number of local minima.

$$f(x) = An + \sum_{i=1}^{n} \left[x_i^2 - A\cos(2\pi x_i)\right]$$

where the $A = 10$, $x_i \in [-5.12, 5.12]$ and the global minimum is $f_{\min} = 0$ at (0,…,0).

4. Schaffer F6:

$$f(x) = 0.5 + \frac{\sin^2\sqrt{x^2 + y^2} - 0.5}{\left(1 + 0.001 \times (x^2 + y^2)\right)^2}$$

where $x_i \in [10, 10]$ and the global minimum is $f_{\min} = 0$ at (0,…,0).

5. Moved axis parallel hyperellipsoid function:

$$f(x) = \sum_{i=1}^{n} 5i \cdot x_i^2$$

where $x_i \in [-5.12, 5.12]$and the global minimum is $f_{\min} = 0$ at (0,…,0).

6. The third Bohachevsky function:

$$f(x) = x_1^2 + 2x_2^2 - 0.3\cos(3\pi x_1) + 0.3\cos(4\pi x_2) + 0.3$$

This function only has two variables, where $x \in [-10, 10]$, the global minimum is $f_{\min} = -0.24$ located in [−0.24,0], [0,0.24].

7. Michalewicz's function:

$$f(x) = -\sum_{i=1}^{d} \sin(x_i)\left[\sin\left(\frac{ix_i^2}{\pi}\right)\right]^{2m}$$

where $m = 10$. $x_i \in [0, \pi]$. When d = 2, the global minimum is $f_{\min} = -1.803$ at position (2.0230, 2.0230).

8. Rosenbrock function:

$$f(x) = \sum_{i=1}^{d-1} \left[(1 - x_i)^2 + 100(x_{i+1} - x_i^2)^2\right]$$

where the global minimum is $f_{\min} = 0$ at positions (1,…1).

## References

1. Özcan E, Basaran C (2009) A Case Study of Memetic Algorithms for Constraint Optimization. Soft Comput Fusion Found Methodol Appl 13(8–9):871–882
2. Yang X-S (2009) Firefly algorithms for multimodal optimization. Stochastic algorithms: foundations and applications, SAGA 2009. Lecture notes in computer sciences, 5792. Springer, Heidelberg, pp 169–178
3. Yang X-S, Deb S (2009) Cuckoo search via Levy flights. In: World congress on nature and biologically inspired computing (NaBIC 2009). IEEE Publication, USA. 2009, pp 210–214
4. Yang X-S, Deb S, Fong S (2011) Accelerated particle swarm optimization and support vector machine for business optimization and applications, the third international conference on networked digital technologies (NDT 2011), Springer CCIS 136, Macau, 11–13 July 2011, pp 53–66
5. Yang X-S (2010) A new metaheuristic bat-inspired algorithm. In: Gonzalez JR et al (eds) Nature inspired cooperative strategies for optimization (NISCO 2010), vol 284., Studies in computational intelligenceSpringer, Berlin, pp 65–74
6. Peng Y (2011) An improved artificial fish swarm algorithm for optimal operation of cascade reservoirs. J Comput 6(4):740–746
7. Törn A, Zilinskas A (1991) Global Optimization. Lect Notes Comput Sci Parallel Comput 17:619–632
8. Golfberg D (1975) Genetic algorithms in search, optimization and machine learning. Addison-Wesley, Reading
9. Kalender M, Kheiri A, Özcan E, Burke EK (2013) A greedy gradient-simulated annealing selection hyper-heuristic. Soft Comput 17(12):2279–2292. doi:10.1007/s00500-013-1096-5
10. Dueck G, Scheuer T (1990) Threshold accepting: a general purpose optimization algorithm appearing superior to simulated annealing. J Comput Phys 90(1):161–175 Elsevier
11. Glover F (1989) Tabu search—part 1. ORSA J Comput 1(2): 190–206
12. Dorigo M, Maniezzo V, Colorni A (1996) Ant system: optimization by a colony of cooperating agents. IEEE Trans Syst Man Cybern Part B 26(1):29–41