10.1109/ICDIM.2012.6360147

# Wolf Search Algorithm with Ephemeral Memory

Rui Tang, Simon Fong
Department of Computer and Information Science
University of Macau
Taipa, Macau SAR
ccfong@umac.mo

Xin-She Yang
Mathematics and Scientific Computing
National Physical Laboratory
Teddington, UK
xin-she.yang@npl.co.uk

Suash Deb
Department of Computer Science & Engineering
C. V. Raman College of Engineering
Bidyanagar, India
suashdeb@gmail.com

***Abstract*—In computer science, a computational challenge exists in finding a globally optimized solution from a tremendously large search space. Heuristic optimization methods have therefore been created that can search the very large spaces of candidate solutions. These methods have been extensively studied in the past, and progressively extended in order to suit a wide range of optimization problems. Researchers recently have invented a collection of heuristic optimization methods 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 proposes a 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 contribution of the paper is twofold: 1. for verifying the efficacy of the WSA the algorithm is tested quantitatively and compared to other heuristic algorithms under a range of popular non-convex functions used as performance test problems for optimization algorithms; 2. The WSA is investigated with respective to its memory requirement. Superior results are observed in most tests.**

***Index Terms*—Metaheuristic; Bio-inspired Optimization; Wolf Search Algorithm**

## I. Introduction

An optimization problem generally aims to find $x^{opt} = \max_{x \in X} f(x)$ 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 $x^{opt}$ that is assumed to exist in the problem space. In many real-life applications, the optimization functions may not behave well mathematically and it is a well-known challenge in searching for a global optimal solution.

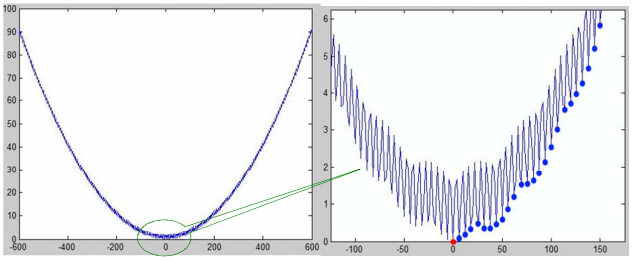

Fig. 1. Griewank Function with it local minima and a global minimum. Left: zoom-out, $x$=[-600, 600], Right: zoom-in, $x$=[-150, 150]

For example, the Griewank function that is shown in Figure 1, has been commonly used to test the convergence of optimization algorithms because its number of local minima grows exponentially as its number of dimensions increases; while its single global minimum is located at $x$=0. In such cases where the global optimum is hard to find, especially when the data carry high dimensional variables, the optimization problems can be complex and the problem sizes may thwart efficient calculation. For instance, in the travelling 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 stochastic optimization. A stochastic optimization is 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. Incrementally, 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 $x^{candidate}$ and as long as the stopping criteria are not met, it checks its neighbors against the current solution (SELECT $x^{neighbor} \in \mathbb{N}(x^{candidate})$ ). The candidate solution is updated with its neighbor if it is better (IF $f(x^{neighbor}) < f(x^{candidate})$ THEN $x^{candidate} = x^{neighbor}$), such that the global optimum at the end is $x^{opt} = x^{candidate}$. As such, heuristic optimization 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 exploration through the solution space is organized. The authors Yang and Deb have recently invented a collection of bio-inspired metaheuristic algorithms, including Firefly [2], Cuckoos [3], Bats [4] and Accelerated PSO [5]. These bio-inspired heuristic optimization algorithms have search methods both in breath and in depth that are largely based on the swarm movement patterns of animals and insects found in nature. Their performance in heuristic optimizations have proven superior to that of many classical metaheuristic methods [9, 10] (e.g. genetic algorithms, simulated annealing, Tabu search, etc).

This paper proposes a new bio-inspired heuristic optimization 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 is in a better position. In this way, long-range inter-communication among the wolves that represent the searching points 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 and 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.

In particular, the memory requirement of WSA is investigated. It is known that wolves in the nature have unparalleled memory capability for they can hide food in caches, they sense (by smelling) and track down a prey from distances of miles away, and they do set markers in their territory by urinating at the borders, etc. Hence wolves are believed to remember their traits well. Biology researchers like Sebastian Vetter and his team, from The Wolf Science Center at the University of Vienna have been studying the observational spatial memory (OSM) in the grey wolf. (http:// http://www.wolfscience.at/english/research/researchers/sebastian_vetter.html). Investigations of OSM in insects and bird species have largely been done, but relatively new it was conducted on mammals. Nevertheless, in the context of computer programming, it is assumed that the wolves that function as searching agents in the WSA optimization algorithm are empowered by memory caches that store the previously visited positions. A contribution as well as the purpose of this paper is to examine how much such memory is needful in WSA for achieving a good solution, in comparison with other similar algorithms.

This paper is organized into the following sections. Section 2 provides a description of the algorithm's logics and operation. Section 3 reports on the experiment and discusses the results. Section 4 concludes the paper.

## II. Description of the Wolf Search Algorithm

### A. Background

Wolves are social predators that hunt in packs. Wolves typically commute as a nuclear family, which is different from PSO and Fish Swarm, which usually move in relatively large groups. Wolves remain silent and use stealth when hunting prey together. Unlike ants, which use pheromones to communicate with their peers about food traits, WSA forgoes this kind of communication, which shortens the run time of the search. Wolves have 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.

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.

### B. The Basic Rules

Based on wolves' hunting behavior, as described above, the three rules that govern the logics of the Wolf Search Algorithm (WSA) are presented as follow.

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 hyper-plane, 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 hyper space. 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 choose 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 chose the best terrain inhabited by another wolf from the given options. Otherwise, the wolf will continue to move 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 Wolf Search Algorithm (WSA) in pseudo code:

```
Objective function f(x), x=(x1,x2,..xd)^T
Initialize the population of wolves, xi(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
α = velocity factor of wolf
pa = 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(xi,xj)<r && xj is better as f(xi)<f(xj))
      xi moves towards xj // xj is a better than xi
    ELSE IF
      xi = Prey_new_food_passively();
    END IF
    Generate_new_location();
    IF(rand()>pa)
      xi = xi + rand() + v; // escape to a new pos.
    END IF
  END FOR
END WHILE
```

Fig. 2. Pseudo code for the WSA algorithm

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 has not been visited by a wolf, given its ephemeral memory. The default 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. As variants of WSA, 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.

### C. Merging with Other Wolves

In the implementation of WSA 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 finite, 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 less than its visual distance. 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; *v*, *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.

### D. 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 Brownian motion because wolves that 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:

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

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 earlier.

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 $f(x(j)) \geq f(x(k))$. $x(j)$ will ignore its nearby peers and move toward the food, which has the highest fitness value. $x(l)$ is roaming in Brownian motion some distance away.

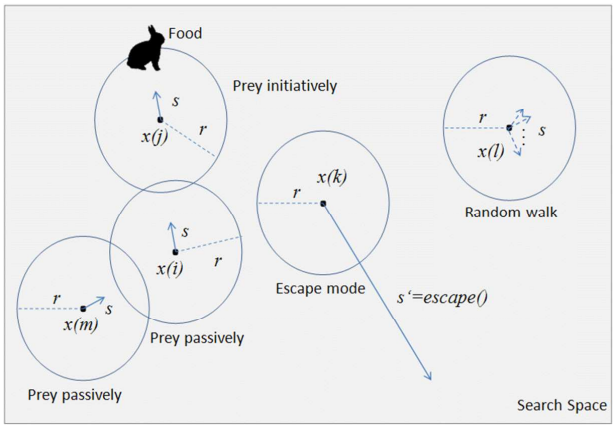


Fig. 3. A snapshot of WSA in action

## III. Experiment

### A. Testing Optimization Functions

To validate this new algorithm, we implemented WSA in MATLAB and tested various famous optimization functions. Here, we use one of them to demonstrate how WSA works. **Schaffer's F6 function** [7] is a non-convex function for testing an optimization algorithm. There are many local minima in Schaffer's F6 function, which makes it highly multimodal. The function is defined as follows:

$$f(x) = 0.5 + \frac{\sin^2\sqrt{x^2+y^2} - 0.5}{(1+0.001\times(x^2+y^2))^2} \quad (4)$$

where $x_i \in [-10, 10]$ and the global minimum is $f_{min}=0$ at (0, .. 0). The visualization of Schaffer's F6 function in 3-D is shown in Figure 4. The global minimum of Schaffer's F6 function is $f(\cdot)=0$ at (0,0). We use $W=20$ wolves and a tolerance factor equal to $\pm 1.0e^{-5}$, under which circumstances WSA easily reaches the minimum. The initial step and final result are shown graphically in Figures 5 and 6, respectively. The blue points are the locations of the wolves. Although this test function has numerous local minima, after running the algorithm almost all of the wolves clearly gather at (0,0), which suggests that WSA deals with multimodal problems pretty well; that is, they are able to find the global optima and converge there.

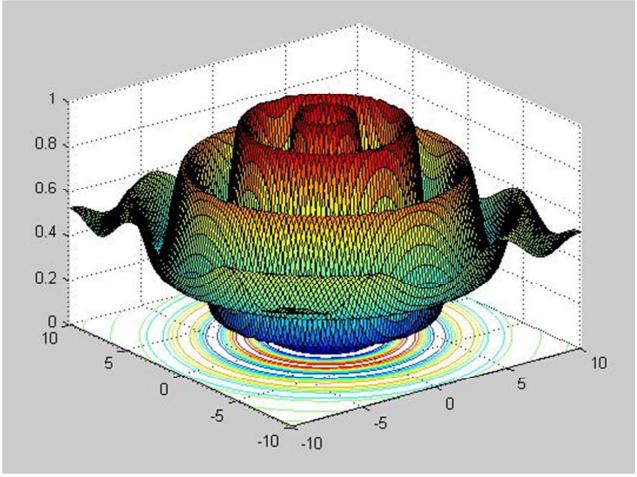

Fig. 4. The visualization of the Schaffer's F6 function

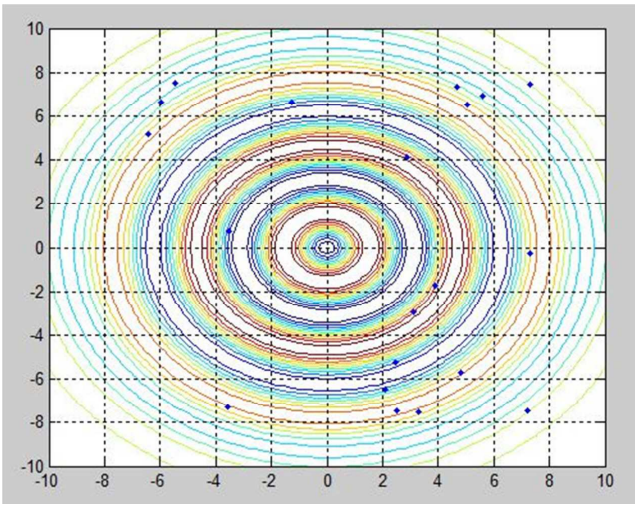

Fig. 5. The wolves' locations at initial step which are distributed at random

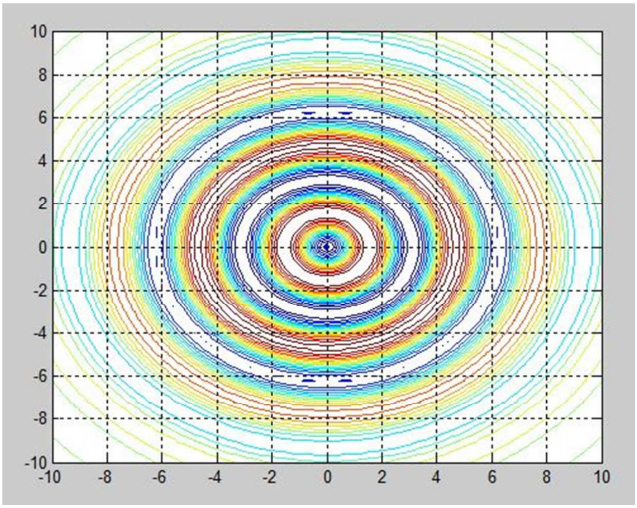

Fig. 6. The wolves' locations at final step, merged at the global minimum

Eight optimization functions are used in the comparison experiment for verifying the efficacy of WSA and other similar algorithms. The mathematical definitions and their global optima are defined as follow.

**Griewangk's function**; in this test function, there are a lot of local minima,

$$f(x)=\sum_{i=1}^{n}\frac{x_i^2}{4000}-\prod_{i=1}^{n}\cos\left(\frac{x_i}{\sqrt{i}}\right)+1 \tag{5}$$

where the data range is $-600 < x_i < 600$, $n$ is user-defined that represents a $n$-dimensional problem, and the global minimum is $f_{\min}=0$ at (0, .. 0).

**Sphere function**, that has no local minimum except the global one,

$$f(x)=\sum_{i=1}^{d}x_i^2 \tag{6}$$

where the search domain is $-5.12 \le x_i \le 5.12$, $i=1,2,..,d$, the global minimum is $f_{\min}=0$ at (0, .. 0).

**Rastrigin's function**; this function is difficult due to its large search space and large number of local minima. This function is often used to test the genetic algorithm.

$$f(x)=An+\sum_{i=1}^{n}\left[x_i^2-A\cos(2\pi x_i)\right] \tag{7}$$

where $A=10$ and $n=2$, the search domain is $-5.12 \le x_i \le 5.12$, $i=1,2,..,n$, the global minimum is $f_{\min}=0$ at (0, .. 0).

**Moved axis parallel hyper-ellipsoid function**, is derived from the axis parallel hyper-ellipsoid. In the end the moved axis parallel hyper-ellipsoid function is more elliptic than the original function.

$$f(x)=\sum_{i=1}^{n}5i\cdot x_i^2 \tag{8}$$

where $n=2$, the search domain is $-5.12 \le x_i \le 5.12$, $i=1,2,..,n$, the global minimum is $f_{\min}=0$ at (0, .. 0).

**The third Bohachevsky function** has two variables.

$$f(x)=x_1^2+2x_2^2-0.3\cos(3\pi x_1)+0.3\cos(4\pi x_2)+0.3 \tag{9}$$

where $n=2$, the search domain is $-10 \le x_i \le 10$, $i=1,2$, the global minimum is $f_{\min}=0.24$ located at [-0.24, 0], [0, 0.24].

**Michalewicz's function** is a multimodal test function. The parameter $m$ defines the "steepness" of the valleys or edges. Larger $m$ leads to more difficult search. For very large $m$ the function behaves like a needle in the haystack (the function values for points in the space outside the narrow peaks offer very little information on the location of the global optimum).

$$f(x)=-\sum_{i=1}^{d}\sin(x_i)[\sin(\frac{ix_i^2}{\pi})]^{2m} \tag{10}$$

where $m=10$, the search domain is $0 \le x_i \le \pi$, $i=1,2,..,d$, the global minimum is $f_{\min}=-1.803$ located at (2.0230, 2.0230).

**Rosenbrock function**; this function only has several local minima.

$$f(x)=\sum_{i=1}^{d-1}[(1-x_i)^2+100(x_{i+1}-x_i^2)^2] \tag{11}$$

where $n=2$, the search domain is $-5 \le x_i \le 10$, $i=1,2,..,n$, the global minimum is $f_{\min}=0$ at (1, 1).

### B. Comparison of WSA to PSO and ACO

Particle swarm optimization (PSO) [4] and ant colony optimization (ACO) [12] are two classical nature-inspired optimization algorithms to which we compared WSA. What PSO and ACO have in common is the inter-agent communication required for the swarm intelligence to work. The basic version of PSO is known to get easily trapped into a local optimum. In order to avoid the premature convergence the best known position of the entire swarm is neglected. Instead, the best known moving position of a sub-swarm "near" the particle is used. The sub-swarm can be geometrically defined as "the nearest particles" or "socially closest particles". To facilitate the topology of the sub-swarm, some information link must exist between each particle and its neighbor particles. Often a graph is established, e.g. a ring topology is used when each particle has only two neighbors. Likewise for ACO, the ants communicate by pheromone. Therefore a pheromone matrix is assumed to store the traits of each ant, each trait represents a solution, and collaboratively the ants update and fine-tune the traits with the strongest pheromone values.

With respect to memory and communication requirements, PSO and ACO could be commonly generalized to a type of heuristic search that requires some intercommunication and the corresponding information links are bound to be properly stored. In contrast, WSA needs no intercommunication during the search. The only memory requirement however may be the extent of previous positions that have been visited by a wolf; hence the old positions are to be avoided when a new position is being generated. Intuitively, the older the positions that the wolf can recall hence avoid, the lesser of these visited positions would be repeated during the search. That may contribute to the efficiency of the heuristic search.

For this reason, two modified versions of WSA are implemented in the experiment, that mimic wolves that have different levels of ephemeral memory – the first variant of WSA, Wolf Search Algorithm 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 positions from all the past rounds of calculations. This implies that wolves possess a temporary memory that remembers not to back-track all the old positions. This variation in meta-heuristics can enhance the search by avoiding a return to old states. WSA-MP obviously requires the maximum amount of memory and memory space for execution. The second variant of WSA, is Wolf Search Algorithm with Step Minus Previous # Steps (WSA-MP#). "#" is a finite integer, arbitrarily chosen by user, which denotes how far the wolf can recall from its memory of its walked places. For example, WSA-MP5 means the wolves in the WSA algorithm can remember up to 5 of the previous visited steps. The memory is constantly updated with fresh values during each round of iteration, and the size of the memory space is limited to 5. The oldest memory is replaced by the most recent information during each move. This assumption of ephemeral memory is realistic as it is known that the memory capacity in mammals is far from infinite.

PSO and the ACO used in this comparison are standard versions with default parameters used for benchmarking purposes. Our experiment uses two approaches to test WSA and the variants. 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 [8] 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 10,000 rounds, regardless of yielded improvements, and then compare the mean of fitness after stop. Each algorithm is repeated 100 times to produce a meaningful statistical analysis. The computing environment is a MacBook Pro (with CPU:2.3GHZ,RAM:4GB).

TABLE I. COMPARISON OF CONVERGENCE CYCLES USING FIXED TOLERANCE THRESHOLD

| Function/Algorithm | PSO | ACO | WSA |
|---|---|---|---|
| Griewangk | 1070 ± 633 | 724 ± 281 | 492 ± 411 |
| Sphere Model | 1670 ± 1024 | 950 ± 328 | 697 ± 798 |
| Rastrigrin | 24206 ± 16000 | 38596 ± 50572 | 18955 ± 18992 |
| Schaffer's F6 | 34046 ± 7461 | 1040 ± 307* | 3404 ± 4809 |
| Moved Axis Parallel | 4779 ± 3186 | 4713 ± 2931 | 3893 ± 5140 |
| Bohachevsky | 3993 ± 3170 | 3124 ± 5231* | 4680 ± 6509 |
| Michalewicz' | 14283 ± 8615 | 5012 ± 3141 | 4988 ± 4721 |
| Rosenbrock | 15197 ± 9554 | 45897 ± 28350 | 6421 ± 7230 |

(continued)

| Function/Algorithm | WSA-MP05 | WSA-MP10 | WSA-MP15 | WSA-MP |
|---|---|---|---|---|
| Griewangk | 459 ± 400 | 412 ± 362* | 422 ± 383 | 425 ± 356 |
| Sphere Model | 538 ± 562* | 639 ± 642 | 672 ± 699 | 677 ± 712 |
| Rastrigrin | 15259 ± 14358* | 20723 ± 19840 | 18370 ± 16728 | 18677 ± 17475 |
| Schaffer's F6 | 4177 ± 6629 | 3605 ± 4205 | 4306 ± 5890 | 4321 ± 6060 |
| Moved Axis Parallel | 4066 ± 4536 | 3165 ± 4356 | 3060 ± 4272* | 4058 ± 6271 |
| Bohachevsky | 4540 ± 4813 | 4195 ± 4579 | 3698 ± 3796 | 4705 ± 5535 |
| Michalewicz' | 4829 ± 4918 | 4821 ± 5029 | 4721 ± 5149* | 5183 ± 5122 |
| Rosenbrock | 5549 ± 5724 | 6128 ± 7971 | 5337 ± 7922* | 6265 ± 6940 |

TABLE II. COMPARISON OF OPTIMIZATION FITNESS USING FIXED NUMBER OF ITERATIVE CYCLES

| Function/Algorithm | GA | PSO | ACO | WSA |
|---|---|---|---|---|
| Griewangk's | 0.1860 | 2.7559e-8 | 1.3836e-10* | 1.9126e-7 |
| Sphere Model | 0.0019 | 9.5715e-8 | 3.7493e-10* | 2.8765e-8 |
| Rastrigrin's | 0.0677 | 1.5829e-5 | 0.0282 | 1.0297e-5* |
| Schaffer's F6 | 0.0090 | 3.8226e-4 | 4.6262e-10* | 1.930e-4 |
| Moved Axis Parallel | 0.0078 | 5.9182e-7 | 3.2337e-9* | 1.8917e-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 | 0.0033 | 2.5192e-6 |

(continued)

| Function/Algorithm | WSA-MP05 | WSA-MP10 | WSA-MP15 | WSA-MP |
|---|---|---|---|---|
| Griewangk | 1.3667e-7 | 1.3553e-7 | 1.5936e-7 | 1.0690e-7 |
| Sphere Model | 2.4068e-7 | 3.1347e-7 | 3.0250e-7 | 1.6620e-7 |
| Rastrigrin | 1.1686e-5 | 1.2615e-5 | 1.1124e-5 | 1.2153e-5 |
| Schaffer's F6 | 9.7097e-5 | 2.2662e-6 | 1.6962e-5 | 9.7579e-5 |
| Moved Axis Parallel | 1.4931e-6 | 2.118e-6 | 1.4889e-6 | 1.8770e-6 |
| Bohachevsky's | -0.2400* | -0.2400* | -0.2400* | -0.2400* |
| Michalewicz's | -1.8011 | -1.8012* | -1.8012* | -1.8012* |
| Rosenbrock's | 3.0540e-6 | 2.8343e-6 | 2.1689e-6 | 6.1665e-7* |

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 generally 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. The superior results are marked with a red asterisk for easy observation in each entry of comparison.

PSO was completely under rated in Table 1. However, the ratio of wins between ACO and WSA is 2:6. WSA outperformed ACO by 3 times of experiment runs. Out of the 6 wins in WSA, (as shown in Table I the lower part) there is an interesting distribution of wins across different variants of WSA algorithms – WSA(0), WSA-MP05(2), WSA-MP10(1), WSA-MP15(3), and WSA-MP(0). Both extreme cases of WSA where the wolves that are assumed to have very short ephemeral memory (just WSA), and the wolves that are assumed to possess huge memory (WSA-MP) score zero win. Relatively, with 3 wins, WSA-MP15 beats WSA-MP05 and WSA-MP10. This phenomenon may suggest that a moderate amount of ephemeral memory (neither too large nor too small) helps the algorithm converges quickest with the least number of runs.

Table 2 demonstrates that GA easily moves into a local minimum that PSO and WSA avoid. Again, ACO and WSA beat PSO in the second experiment. Moreover, ACO scores 6 wins out of 8 runs that show ACO achieves more accurate results than the rest. Given a fixed number of iterative cycles in the experiment runs, the optimization algorithms were yet to converge – therefore the experiment rather shows a premature or intermediate stage of optimization. ACO yields superior results mainly because the optimization ACO is incremental in nature. Its pheromone matrix that represents the best routes clued by the ants gets improved progressively in each run. Though many repetitions are required for convergence (as shown in Table 1), the intermediate results of optimization are outstanding. This shows a scenario where the optimization result gets paid off at the cost of the large number of evaluations. For WSA, when the optimization has not converged, the extreme cases of WSA memorizing only one previous step and WSA memorizing all the previous steps, produce relatively good results.

### C. Discussion

Heuristic optimization can be guided or unguided. 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 ACO represents the adaptive memory of previously visited solutions; and the Tabu list provides short term memory [11]. 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 certain previous steps (solutions) are 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. As shown in our experiment results, remembering the past trail to a certain length provides better performance at a moderate cost of memory and time. Contrary to common belief, a full length of past trait may not always help in attaining the best results.

Moreover, WSA uses no amount of information for inter-communications. This may be an advantage for some application. Ants 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 Ant Colony 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.

## IV. Conclusion

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 [13] and their applications. For instance, clustering integrated with nature-inspired optimization produces improved performance [13]. 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 heuristic optimization algorithm, the Wolf Search Algorithm (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. WSA is tested against classical algorithms such as GA, PSO, ACO, and it outperformed GA and PSO in most of the testing cases, and beats ACO in the convergence test. In this paper, we tested particularly the memory requirement for WSA as to find out how much the previous trait should be remembered so as to avoid them in generating new solution. The results indicated that a moderate amount of previous steps should be memorized for obtaining good results. The WSA's potential contributions to finding optimal solutions in applications include but are not limited to the following: Travelling salesman problems, Quadratic assignment problems, Job scheduling problems and Sequential ordering problems.

## REFERENCES


[1] M. Gilli, P. Winker, "A Review of Heuristic Optimization Methods in Econometrics", Swiss Finance Institute Research Paper No. 08-12. June 2008, Available at SSRN: http://ssrn.com/abstract=1140655.

[2] X.-S. Yang, "Firefly algorithms for multimodal optimization". Stochastic Algorithms: Foundations and Applications, SAGA 2009. Lecture Notes in Computer Sciences. 5792. pp. 169–178.

[3] X.-S. Yang, S. Deb, "Cuckoo search via Levy flights", in: World Congress on Nature and Biologically Inspired Computing (NaBIC 2009). IEEE Publication, USA. 2009, pp. 210–214.

[4] X.-S. Yang, S. Deb, S. Fong, "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, 11-13 July 2011, Macau, pp.53–66.

[5] X.-S. Yang, "A New Metaheuristic Bat-Inspired Algorithm", in: Nature Inspired Cooperative Strategies for Optimization (NISCO 2010), Eds. J.R. Gonzalez et al., Studies in Computational Intelligence, Springer Berlin, 284, Springer, pp. 65–74.

[6] Y. Peng, "An Improved Artificial Fish Swarm Algorithm for Optimal Operation of Cascade Reservoirs", Journal of Computers, VOL. 6, NO. 4, April 2011, pp.740–746.

[7] A. Törn and A. Zilinskas. "Global Optimization", Lecture Notes in Computer Science, Parallel Computing, 17, 1991, pp. 619–632.

[8] D. Golfberg, "Genetic Algorithms in Search, Optimization and Machine Learning", Addison-Wesley, Reading, MA 1975.

[9] S. Kirkpatrick, C.D. Gelatt, M.P. Vecchi, "Optimization by Simulated Annealing", Science 220 (4598), 1983, pp.671–680.

[10] G. Dueck, T. Scheuer, "Threshold Accepting: A General Purpose Optimization Algorithm Appearing Superior to Simulated Annealing", Journal of Computational Physics, Volume 90, Issue 1, Elsevier, September 1990, pp.161–175.

[11] F. Glover, "Tabu Search - Part 1", ORSA Journal on Computing 1 (2), 1989, pp.190–206.

[12] M. Dorigo, V. Maniezzo, A. Colorni, "Ant System: Optimization by a Colony of Cooperating Agents", IEEE Transactions on Systems, Man, and Cybernetics–Part B, 26 (1): 1996, pp.29–41.

[13] R. Tang, S. Fong, X.-S. Yang, S. Deb, "Integrating Nature-inspired Optimization Algorithms to K-means Clustering", Proceedings of 7th International Conference of Digital Information Management (ICDIM 2012), IEEE, August 2012.