IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received September 8, 2017, accepted December 10, 2017, date of publication January 23, 2018, date of current version March 15, 2018.

Digital Object Identifier 10.1109/ACCESS.2018.2797059

Next Generation Metaheuristic: Jaguar Algorithm

YAO-HSIN CHOU, SHU-YU KUO, LI-SHENG YANG, AND CHIA-YUN YANG
Department of Computer Science and Information Engineering, National Chi Nan University, Puli 545, Taiwan
Corresponding author: Yao-Hsin Chou (yhchou@ncnu.edu.tw)

ABSTRACT Metaheuristic algorithms are implemented to solve optimization problems and have recently received significant research attention. Metaheuristic algorithms rely primarily on two properties, exploration, and exploitation. Traditional metaheuristic algorithms use many weights (parameters) to balance these two properties to increase the chance of finding a better solution in limited cost and time. However, traditional algorithms have some problems. Exploration and exploitation are different abilities and restrict each other, therefore, traditional algorithms need many parameters and lots of costs to achieve the balance, and also need to adjust parameters for different optimization problems. Jaguar Algorithm (JA) has great abilities both in exploitation and exploration, is proposed to address these issues. First, JA attempts to find the optimal solution in the designated search area. It then uses history information to jump to a better area. JA can, therefore, determine the position of the global optimum. JA achieves strong exploitation and exploration with these features. Also, according to different problems, JA implements adaptive parameter adjustment. The self-analysis and experiment of this research demonstrate that each JA capability can have various positive effects, while the performance comparison demonstrates JAs superiority over traditional metaheuristic algorithms.

INDEX TERMS Metaheuristic algorithms, jaguar algorithm (JA), function optimization problem, dependency problem.

I. INTRODUCTION

As modern life grows ever more complex in all areas, the need for efficiency and methods for reducing resource consumption has become more important. Many situations in finance or manufacturing, for example, require the resolution of highly complicated optimization problems. Examples include stocks selection [1] and finding trading rule [2], [3] in stock markets, as well as circuit synthesis, placement and routing optimization problems, and deployment optimization problems [4] in wireless sensor networks and some traditional problems include the 0/1 knapsack problem, scheduling problems, traveling salesman problem (TSP), etc. These have been proven to be NP-hard/NP-complete problems, and they are important problems need to be solved efficiently and effectively. Effective algorithms are therefore required to find optimal solutions in a short time. Generally, the solution space is extremely large, and a plurality of variables or dimensions must be considered. To identify the best quality solution in a large solution space, an algorithm must test many variable combinations to find the best one. However, it is very difficult to find the optimal solution from these numerous combinations. This means that the optimal solution cannot be found by the method of exhaustion in a limited time; it is thus important to develop an effective and efficient algorithm for this purpose. Manny metaheuristic algorithms have been proposed to solve these highly complicated optimization problems. On the condition of limited time and cost, metaheuristic algorithms help to find the approximate optimal solution from the huge solution space in a limited time. In the real world, the goal of metaheuristic algorithms is to minimize cost and maximize profit in regard to industrial and business problems. The better the capability of an metaheuristic algorithm, the better the solution will be, and found in a shorter time. For this reason, metaheuristic algorithm has received global research attention.

Heuristic methods are mainly used to solve specific problems, while metaheuristic algorithms can be applied to solving different types of optimization problems. Most of metaheuristic algorithms are inspired by observing the behavior of animals or natural phenomena. Different types of metaheuristic algorithm have been proposed with promising performance, such as genetic algorithm (GA) [5], differential evolutionary (DE) [6], [7], evolution strategy (ES) [8], genetic programming (GP) [9], particle swarm optimization (PSO) [10], ant colony optimization (ACO) [11],

artificial bee colony (ABC) [12], quantum-inspired evolutionary algorithm (QEA) [13], quantum-inspired electromagnetism-like mechanism (QEM) [14], quantum-inspired tabu search (QTS) [15], [16], and tabu search algorithm [17]. These metaheuristic algorithms use different information to search the solution as thoroughly as possible. Recently, swarm algorithms constitute an emerging branch of metaheuristic algorithms because they are simple and effective. Swarm algorithms are population-based algorithms that simulate the social behavior of swarm animals such as an ant colony, bee colony, a flock of birds, a school of fish, or an immune system. Swam system collects information from each individual in the group. In this study, we propose a novel metaheuristic algorithm, named jaguar algorithm (JA), which is inspired by the behavior of jaguars. JA is a state-of-the-art algorithm that enhances the ability of exploration and exploitation and can significantly improve the premature convergence problem which traditional swarm algorithms get trapped into a local optimum easily when solving multimodal problems.

Exploration is dedicated to looking for promising areas within the search space, while exploitation focuses on these promising areas and searches them thoroughly. The traditional metaheuristic search process attempts to achieve a balance between exploration and exploitation that is an important research focus in the field of metaheuristic algorithm. By striking a better balance between exploration and exploitation, the algorithm can achieve a better searching capability. However, there are some problems in search methods attempting to balance these two forces. It is not easy to adjust these two forces because they are contradictory to each other. If the algorithm over-concentrates on exploration, convergence becomes difficult and hard to find global optimal. If the algorithm over-concentrates on exploitation, it will cause the problem of premature and can be trapped in local optima easily. These algorithms thus require many parameters to balance exploration and exploitation when dealing with different types of problems. Algorithms also still need to test all the parameter combinations in order to find the optimal solution.

JA differs from traditional search methods. JA conducts exploration and exploitation separately, which means that it does not need to balance the two forces. JA focuses on exploitation first, discovering the optimum in a local area as far as possible. Then, it uses territoriality to prevent other jaguars or itself entering those searched areas again. The mechanism of territoriality makes jaguar go to areas not previously searched, thus preventing extra evaluation, and reducing resource consumption. Finally, JA uses the information of all territories to jump to different places (i.e. exploration). With the above schemes, JA exploits promising areas and can explore territories without becoming trapped in a local optimum. JA can, therefore, perform well in terms of both exploration and exploitation. In this paper, This paper uses various function optimization problems to test the searching ability of JA. The results presented here indicate that the JA outperforms the other traditional methods.

The remainder of this paper is organized as follows. SectionII briefly introduces the related literature on metaheuristic algorithms, discusses the major search methods of these algorithms, and identifies contributions and deficiencies. SectionIII presents the process and the characteristics of JA, which offers a fundamental improvement. SectionIV provides numerical comparisons with the related studies in order to test the capabilities of algorithms by function optimization problems and analyze the performance of JA. SectionV offers conclusions for this research.

II. RELATED WORK

Many metaheuristic algorithms studies have been conducted to date, focusing on solving highly complicated optimization problems in different fields. This section will discuss some state-of-the-art metaheuristic algorithms. They all have their own characteristics; some of them are inspired by the behavior of animals, and some by natural phenomena. Each metaheuristic algorithm has its advantages and disadvantages. The following will discuss the search ability of these algorithms.

Evolutionary algorithm is inspired by the natural rules of survival and natural selection. In evolutionary terms, the species with the best genes survive, and pass those genes to the next generation, which undergoes the same selection process. As a result, only the fittest species will survive. GA [5] is the most famous in this category. The primary processes in GA are cloning, crossover and mutation. Each solution is represented by one chromosome. Better chromosomes leave better information through cloning, while inferior ones will be eliminated. Parents preserve good parts through a crossover operation, and change parts of the chromosomes by mutation. By repeating these processes, chromosomes can converge to a better solution, i.e. survival of the fittest. However, GA suffers from premature convergence because of the cloning and crossover operations. This means that solutions can be easily trapped in local optima. So GA needs mutation to escape local optima. But too low a mutation rate could mean not finding a better solution, and too high a mutation rate causes algorithm divergence. Thus, GA needs an appropriate cloning method, a suitable crossover method, and a fit mutation rate in order to have better search capabilities.

Swarm algorithms are mostly inspired by the behaviors and habits of social animals in nature. They usually have their own means of communicating, so they are able to find targets in a short time. One of the more popular algorithms is PSO. PSO is inspired by the behavior of birds flocking. Many different methods have been designed to improve the performance of PSO and analyze its search ability [10], [18]–[20]. PSO merges a current global best solution (*gbest*), personal best solutions (*pbest*), and inertial velocity (v) into a resultant force. The resultant force leads the swarms to search promising areas, so PSO needs a good set of velocity weights and inertia weight ($c1$, $c2$ *and* w) in order to look around the domain so as to converge in a better place. Because of its

simple implementation and good quality of solutions found, PSO can be applied to many fields. Many variant versions of PSO attempt to enhance the quality of solutions and solve more complex problems. ABC algorithm [12], [21] is another famous algorithm in this category. ABC mimics the behavior of honey bee swarms, and includes the division of labor and cooperation in searching for and collecting nectar. The main feature is that the swarm population is divided into three kinds of characters: employed bees, onlookers and one scout. Each of them has its own duty. Employed bees are responsible for searching around food sources. The onlookers get information about food sources and choose a target by roulette wheel selection. Only the scout is similar to the mutation operator in GA. It moves randomly in the search space. If a food source cannot be improved further through a given number of iterations, called the *limit*, it is replaced with a new food source by the scout in order to escape from the local optima. In ABC, employed bees and onlookers are in charge of the exploitation in the domain, and scouts are responsible for exploration and dispersion to search for potential food sources. ABC lacks a better exploration capability. It therefore uses numerous populations and increasing iterations to solve the problem, but the evaluation also increases. It is necessary to adjust the appropriate number of employed bees, onlookers, scouts and the *limit* in order to avoid becoming trapped in local optima. This is how ABC balances exploration and exploitation. Other swarm algorithms such as ACO [11], artificial fish swarm algorithm [22], grey wolf optimizer [23], bat algorithm [24], glowworm swarm optimization [25], etc., are similar to PSO, and combine different parameters into a resultant force. Some also add a random factor to seek other solutions. Some algorithms implement a local search, but this results in many wasted evaluations if the location is not near the best solution.

QEA [13], proposed in 2002 by Han and Kim was the first quantum-inspired algorithm. Subsequently, many researchers have proposed different quantum-inspired algorithms or improved the original, resulting in, for example, QEM [14], QTS [15], [16], improved quantum-inspired tabu search algorithm (IMQTS) [26], and Quantum-inspired particle swarm optimization (QPSO) [27]. QEA mimics the superposition phenomenon, whereby a quantum group represents all possible states at the same time. The state of a quantum bit has a probability to be 0 or 1 through measurement. QEA generates populations *P(t)* by measuring the probability matrix *Q(t)*. The better population will then be stored in *B(t)*. The best fitness in *B(t)* is denoted as the best solution *b*. A migration in QEA is implemented by replacing some solutions by others, locally or globally. QEA adjusts the probability matrix *Q(t)* by Q-gate. The idea of QEA is to get closer to the best solution. The angle of the Q-gate is designed according to different problems. Too large an angle causes the solution to become stuck at local optima, and too small an angle causes solution divergence. QTS uses the same method as QEA to generate populations. The difference between QTS and QEA is that QTS updates the beta array with a rotation gate according to both the best and worst solutions. The probability of the corresponding beta array is increased or decreased according to the best solution and the worst one. The idea of QTS is to get away from the worst, and closer to the best solutions. Also, the angle of the rotation matrix θ_1, θ_2 affects its optimization capability. This is why QTS needs to be adjusted for suitable parameters to get the best solution. IMQTS tends to solve the problem of QTS, which is unable to escape local optima, to search for the global optimum. IMQTS combines a bell curve probability model $f = (x;\ a,\ b,\ c)$ into QTS. The bell curve probability model provides IMQTS with a certain probability to accept the second best solution instead of the best solution. It is therefore easier for IMQTS to escape the local optima, and thus able to explore the solution space more effectively. However, quantum-inspired algorithm needs many generations to adjust the beta array with the angle of rotation array to avoid converging to local optima too fast, or it employs a jump mechanism to increase the probability of getting the best solution.

Tabu search (TS) algorithm [17] is a very special algorithm that is usually employed to solve combinatorial optimization problems. The concept is to avoid trying the same solutions or patterns in order to find better solutions. Tabu search algorithm has the ability of memory, and recent solutions that have been tried are recorded in a ''tabu list''. If the tabu list is full, the elements expire from the tabu list in the same order they were added. TS searches neighbor solutions with a hill climbing heuristic and avoids positions which are in the tabu list. The purpose of this algorithm is to escape local optima.

Each of these traditional metaheuristic algorithms has its own search characteristics but mainly attempts to balance exploration and exploitation. At the beginning of the algorithms, the positions of individuals are randomly initialized over the search space; this is an explorative search. During the search process, individuals find the best current solution and will be regarded as the best individuals. This attracts other individuals to exploit the promising area. The exploitation process would be, for example, a superior chromosome in GA being retained for crossover and cloning. In PSO the positions of *gbest* and *pbest* are the search directions, the onlooker bees of ABC exploit the surrounding area of a food source, QEA adjusts the probabilities to get closer to the best solution, QTS brings the probabilities of the beta array closer to the best solution, and farther from the worst solution at the same time. Furthermore, in traditional metaheuristic algorithms, the moving process of other individuals is added by some random factors for the purpose of looking for promising areas; this takes the explorative search into account. Examples would be the mutation rate of GA, the random factors of PSO (*r1, r2*), the random factor of ABC (φ), and the probability matrix of QEA and QTS.

However, it is difficult to focus on exploration and exploitation at the same time. During the process of exploration, algorithms will search as far as possible, so it is difficult to carefully search one area, which is why it is difficult

to simultaneously work on exploitation. Conversely, during the process of exploitation, algorithms will search within promising areas as far as possible, so it is difficult to also search more areas at the same time, which means it is difficult to simultaneously work on exploitation. Traditional search methods thus try to simultaneously explore more promising areas and exploit the current best promising area. Too strong an explorative search might lead to later convergence, or it could result in a failure to converge at a global optimum, wasting evaluations on inferior areas. Too strong an exploitative search might lead to premature convergence, and the algorithm would become trapped in local optima. Therefore, traditional methods try to balance exploration and exploitation, using many parameters to adjust the balance. Examples of this would be the mutation rate and the number of crossovers and clones in GA, and the weight parameters of guidance (inertial velocity, *pbest, gbest*) are respectively *w, c1, c2* in PSO. These weight parameters balancing exploration and exploitation have to be tested repeatedly. Fine-tuning the appropriate parameter combination is a difficult and complicated process. Moreover, in order to solve different complicated problems, different tailor-made parameter combinations must be fine-tuned for each problem. In addition, the movements of traditional methods usually include some random factors. They are thus unable to determine if the exploitation is finished or not, and they have no idea whether they should continue exploiting or jump out to other areas. They need analysis to know whether they are occupied in exploitation or exploration. In addition, other individuals are attracted to exploit promising areas, and they randomly look anywhere where better solutions may be found at the same time. This means that traditional methods are unable to achieve their best possible exploration and exploitation simultaneously.

Unlike these traditional methods, Jaguar Algorithm (JA), based on the behavior of jaguars in the wild, is designed with a completely new concept. Instead of the traditional method of conducting exploitation and exploration at the same time, JA performs these processes separately. It can, therefore, engage in exploration and exploitation at the right moment for each. In JA, the jaguar (algorithm) focuses on exploitation and then focuses on exploration. It pays attention to finding the best solution in the current area when hunting, which means hunting gives JA a strong exploitation capability. Furthermore, territoriality prevents jaguars from searching the area (territory) again if it has been occupied by other jaguars or itself. After this, JA uses the territory information in its memory (tendency) to jump into new areas with better prey (solutions). By jumping, JA avoids becoming trapped in local optima and is able to find better solutions more efficiently. In other words, JA achieves a strong exploration capability by jumping. JA also implements an adaptive adjustment method to approximate parameters according to different problems. Therefore, JA is able to find the best solution much more effectively and efficiently.

III. PROPOSED METHOD

Jaguars are a species of big cat. They are powerful and agile hunters. When a jaguar finds its prey, it moves directly towards the prey in a short time. Then, when it is very close to the prey, it advances carefully rather than attacking recklessly. Jaguars usually hunt alone and consolidate their own territories to prevent competition for prey by other jaguars, which is territoriality. Jaguars only act with other jaguars when they are cubs and learn how to seize prey. When they are mature and strong enough, they leave their parents and establish their own territories. The proposed algorithm, jaguar algorithm (JA), mimics the behavior of a jaguar hunting, learning, and territorial claims. Hunting and learning help jaguars find the nearest prey (solution) which can enhance the capability of exploitation. With territoriality, a jaguar claims its territories to prevent other jaguars from competing for prey. The territories will not be searched again. After this, the jaguar uses the information about its territories to jump to other regions which have not been searched. The territories use memory structures to enhance the capability of exploration. In other words, with these behaviors, JA has strong exploitation and exploration abilities and JA has powerful ability to find the optimal solution.

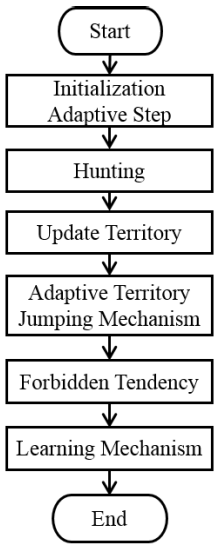

FIGURE 1. Flowchart of jaguar algorithm.

The flowchart of JA is shown in Fig. 1. In initialization, *step* would be set according to the domain of function problems. In each dimension, it executes hunting, updating territory, adaptive territory jumping mechanism, forbidden tendency and leaving forbidden tendency, until the forbidden range is larger than the domain. JA repeats these procedures until the last dimension is reached. After this, JA hunts in each dimension, respectively, to ensure that the jaguar remains in the best territory position. At the end of JA, learning is implemented from the first dimension to the last. The following subsections will discuss each movement of JA clearly.

A. HUNTING

The optimal solution is usually surrounded by other good ones. In order to find the prey (best solution), jaguars should find the best direction, follow the direction and run to the prey quickly. In each dimension, the directions only consist of the positive (+) and the negative (−). At first, the jaguar tries these two sides of the current position to find the best direction ($X \pm step * 2^m, m = 0$ and *step* is the only adaptive parameter in JA. The adaptive step is described in the next subsection). It compares the fitness values with $f(x)$ and chooses the best one as the potential direction. If neither direction is better than the current position, the jaguar remains at its current position. Otherwise, JA doubles the speed (adding m by 1) repeatedly and follows the direction until the fitness of the next position is bad. By a double speed, the jaguar can get close to the prey (solution) rapidly. When a bad position is encountered, JA stops adding m. This means that the jaguar is close to the prey (solution) and needs to move carefully. After this, JA subtracts m by 1 and checks its position in relation to the previous direction which has already been checked. In order to get closer to the prey, JA keeps subtracting m by 1 (slow down the movement), trying on both sides of its current position ($X \pm step * 2^m$) and replacing the current position by the better one until m is equal to 0. As m equals 0, the jaguar is nearest to the prey at the current step. The example of hunting mechanism is illustrated in Fig. 2.

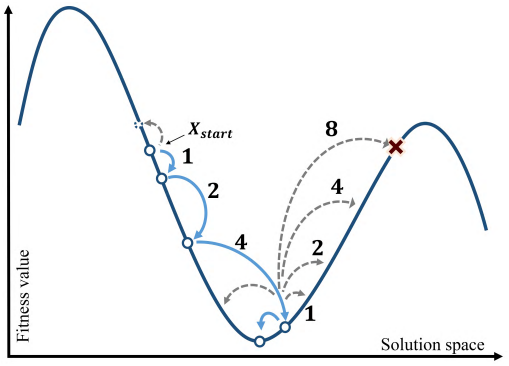

FIGURE 2. Hunting.

JA keeps dividing the *step* by 2 and starting the next round of hunting until the local optimal solution in the current peak is found. When the step is too small that the current position is equal to both sides ($X \pm step = X$, because of the limitation in IEEE754 [28]), the jaguar is probably in the best position for the current peak. In other words, the jaguar has seized its prey. The hunting process pseudocode is shown in Fig. 3. The Rush and Approaching pseudocode is shown in Fig. 4.

For example, in Fig. 2, ***Stage 1***: As hunting begins, the position where the jaguar initialized is denoted as X_{start}. The jaguar tests both sides of this position to select the direction of the next step. If the next position is better, the blue line shows the movement. If the next position is not better than the current position, the gray dotted line shows the movement;

```
Hunting
Input: X
Set step = initial step
  While(X + step != X and X - step != X)
  Rush And Approach(X, step);
  step /=2.0;
End while
claim or update territory with position X, r = |X_start - X| for current dimension
```

FIGURE 3. Hunting pseudocode.

```
Rush And Approach()
Input: X, step
test two sides of X with step, X_L = X - step; X_R = x + step;
If f(X_L) or f(X_R) is the best
  dir = -1 or 1;              //- or + is the best direction
  X = Best(X_L, X_R);
  initial m = 1;
  do
    X_next = X + step * dir * 2m;
    If X_next == X             //IEEE754
      Break;
    Else If f(X) is better than f(X_next) Try same direction once
      m--;
      X_next = X + step * dir * 2m;
      X = Best(X_next, X);
      m--;
    Else if X_next is better than X
      X = X_next;
      m++;
    End if
  while X is updated
  While m >= 0 and X != X ± step * 2m
    X_L = X - step* 2m; X_R = X + step* 2m;
    X = Best(X_L, X, X_R);
    m--;
  End while
End if
Return X;
```

FIGURE 4. Rush and approaching pseudocode.

Stage 2: Because the right side is the best of those three positions, the jaguar follows the direction and adds m by 1 (acceleration) until the fitness of the next position is bad. The brown X marks the last position of acceleration. This means that the jaguar needs to slow down. When $m = 3$, the fitness of the next position is bad. The jaguar keeps subtracting m by 1 and trying both sides of its position until it finishes approaching the prey and $m = 0$; ***Stage 3***: After several rounds of hunting, and when the step is too small to distinguish both sides from the current position, the fitness of the current position is the best in the hunting process, and the position is recorded as X_{end}. When the jaguar finds the prey (local optimum), it will establish a territory with the best solution (X_{end}) and initial position (X_{start}). After the territory is claimed, the jaguar leaves the territory to seek more prey.

B. ADAPTIVE INITIAL STEP

The purpose of the hunting process is to find the best solution of the current peak. In hunting, the jaguar follows the best

direction to find the best solution. If it moves too slowly, the jaguar is able to check details, but will take a long time to reach the prey. Conversely, if it moves rapidly, the jaguar will lose too much information. A suitable step can efficiently find the best solution. In the proposed method, the step is adaptive to different situations. In the beginning, the step is initialized according to the domain range of the solution space or the current position in Equation 1. These are known in the beginning to produce the initial step. JA is then able to obtain the max and min exponent of every value according to IEEE754 [28]. For example, the assumed value is 100 (the domain value of the function problem or the first position of the jaguar), and it is converted into binary representation, which is 1100100, since the floating storage form of IEEE754 [28] is a binary representation. The maximal bit of 1100100 is 2^6. As with IEEE754 [28], the minimal storage bit is 23 bits apart from the maximal bit, so the maximum and minimum of 2^{power} are 2^6 and 2^{-17}. The exponents exp_{max} and exp_{min} are 6 and -17, respectively. In this case, the initial step is 2^{-5}. The formula design will be explained in the experiment section. The jaguar follows the best direction and accelerates the step in hunting. When both sides of the current position are bad, the step will be divided by 2. In the following section, several experiments are conducted to test the capability of the adaptive step. In JA, Adaptive Initial Step calculated with a domain is implemented.

$$step_{ini} = \begin{cases} 2.0^{\left\lfloor \frac{exp_{max}+exp_{min}}{2} \right\rfloor} & if \ \frac{exp_{max}+exp_{min}}{2} \geq 0 \\ 2.0^{\left\lceil \frac{exp_{max}+exp_{min}}{2} \right\rceil} & if \ \frac{exp_{max}+exp_{min}}{2} < 0 \end{cases} \quad (1)$$

C. TERRITORIALITY

Jaguars live and hunt alone for the most part. Each jaguar has its own hunting area, or territory. It does not allow other individuals to compete for the same prey. JA mimics this behavior. In JA, a jaguar claims its territory after finding the prey. Through hunting, the jaguar does its best to find the optimal solution of the current area. Therefore, a territory only consists of the location of the best solution (X_{end}) and the hunting range (r), shown in Fig. 5. The location of the best solution is equal to X_{end} and the hunting range (r) is equal to $Distance(X_{start}, X_{end})$ which is the distance between initial position (X_{start}) and best solution (X_{end}). Establishment of territory means that an area has already been searched, and the best solution of the current area has been found. The idea of territoriality also means that other jaguars will not enter established territories. Therefore, the jaguar is able to search undiscovered areas by utilizing the information of territory, making the force of exploration more efficient.

It is worth mentioning that the idea of territory is similar to the Tabu Search Algorithm [17]. Tabu is well-known for being implemented to solve combinational optimization problems. TS, however, can potentially choose the same solution; however, traditional algorithms cannot record all their positions owing to the limits of storage. In JA, the jaguar will not search those areas (territories) that have already been searched. Territory only consists of the best position (X_{end}) and the hunting range (r). The algorithm attempts to find the best solution in every area until it reaches the optimum of that area. This means that jaguars randomly placed in a territory will locate the same best solution (X_{end}) through hunting. For this reason, the information of territory contains only the best solution and hunting range. This also means that JA does not need to store all searched positions. Traditional methods must record a huge number of positions in a memory pool. Implementation of the proposed method is far more efficient. This territory mechanism also prevents the algorithm from searching the same regions more than once, reducing unnecessary evaluations.

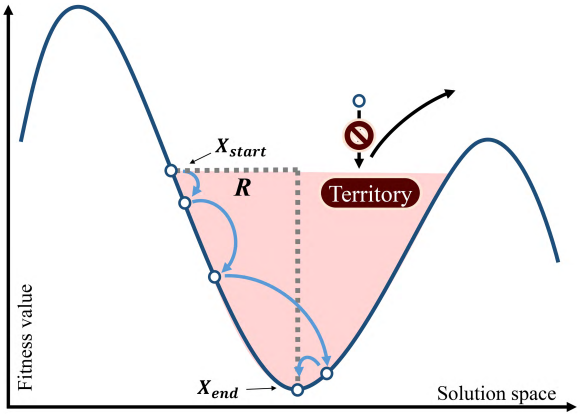

FIGURE 5. Territory.

D. ADAPTIVE TERRITORY JUMPING MECHANISM

Establishing territory means that the best solution of the current area is found. Therefore, JA is able to search other regions which have not been discovered. However, the method for efficiently jumping to new areas is very important. There are some characteristics worth mentioning. The solution space is continuous, if JA follows the good areas, it will be able to find more areas which are much better. In other words, JA is able to find the best area by following the tendency of good areas. Therefore, JA follows the tendency for efficient jumping. The jumping mechanism is similar to the hunting process. First, the jaguar tries to find the tendency, or the good direction, to jump. The jaguar then follows the tendency and accelerates until it finds a bad territory. The jaguar then decelerates and jumps to both sides of the current territory.

1) ADAPTIVE TERRITORY

The first step of jumping is to find the tendency. After hunting, the jaguar claims a territory (t_{cur}). In other words, the information on the current area has been acquired: best solution (fit), location of best solution (X) and hunting range (r, range of current area), as shown in Fig. 6. First, JA tries the

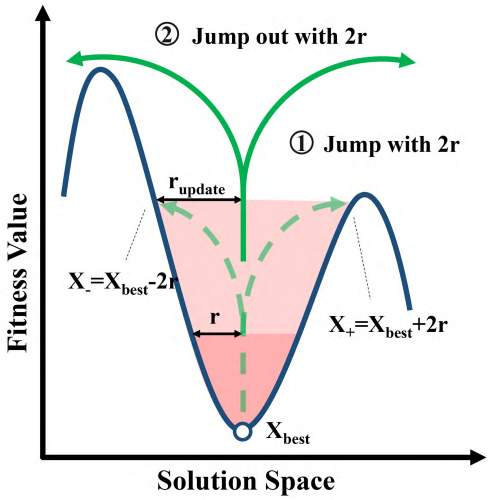

FIGURE 6. Adaptive territory.

two sides of the best solution with the double hunting range ($X_+ = X_{cur} + 2 \times r$; $X_- = X_{cur} - 2 \times r$; *jumping distance* $= 2 \times r$) to jump from the current territory. Then it starts to hunt from these two positions. Due to the hunting process, if the jaguar stays in the same territory, it will find the same best solution (fitness $= fit_{cur}$ and location $= X_{cur}$). If the jaguar cannot jump out of the current territory, the green dotted line shows the movement. It is able to realize that the range of the current territory is larger than the previous one. In this case, the hunting range (r) of the territory will be updated. It will keep trying to jump out until it finds new territories (right side: t_+, left side: t_-) which contain their best solutions, location and hunting range (right side: $fit_+, X_+, r_+ = Distance(X_{cur}, X_+)$; left side: $fit_-, X_-, r_- = Distance(X_{cur}, X_-)$). Alternatively, it could reach the bound of the solution space. If the jaguar jumps out of a territory and finds a nearby territory, the green solid line shows the movement. By this process, it is able to find an appropriate range of a territory. The pseudocode for this process is shown in Fig. 7.

2) ACCELERATION

After the previous process, two new territories besides the original (t_{cur}) are identified. If either of them is better than the current territory, the jaguar will move to the best one in (t_+, t_-) and follow the best direction. If the found territory is better than the previous one, the green solid line shows the movement. If the found territory is not better than the previous one, the green dotted line shows the movement. Then, JA uses the distance between locations of the best territory (from t_+, t_-) and the original one as *jumping distance*. For example, as in Fig. 8, if the right direction (+) is better, move to the position of X_+ and jump again in the same direction by double the jumping distance $X_{next} = X + d \times 2^n$, *jumping distance* $d = Distance(X_{cur}, X_+)$, where n is initialized to 1. If the fitness of the new territory t_{next}

```
Adaptive Territory Jumping Mechanism
Input: t (current territory that jaguar stays)
Do
  If not finding a new territory in the Right side and not reaching upper bound
    X_R = t.X + 2 * t.r;
    If X_R >= UpperBound
      X_R = UpperBound
    End if
    Hunting(X_R);
  End if
  If not finding a new territory in the Left side and not reaching lower bound
    X_L = t.X – 2 * t.r;
    If X_L <= LowerBound
      X_L: = LowerBound
    End if
    Hunting(X_L);
  End if
While not finding new peak and not reaching bound in each side
If t_L or t_R is the best
  dir = -1 or 1;              //- or + is the best direction
  DIST = |X- Best(X_L, X_R)|;
  t = Best(t_L, t_R);
  initial n = 0;
  do
    X_next = t.X + DIST * 2n * dir
    If X_next is out of bound
          Set X_next on the bound
    End if
    Hunting(X_next);
    if t_next is better than t
      t = tnext;
      n++;
  While find a new better territory
End if
Do JumpingDistanceReducing(DIST, n, t)
```

FIGURE 7. Adaptive territory jumping mechanism pseudocode.

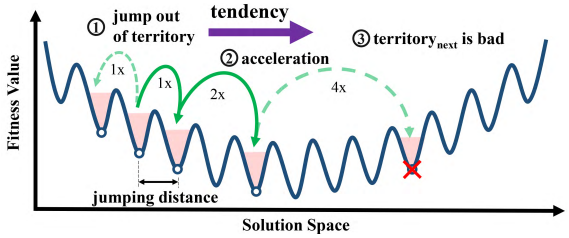

FIGURE 8. Acceleration of adaptive territory jumping mechanism.

(information includes $fit_{next}, X_{next}, r_{next} = Distance(X_+, X_{next})$) is better than that of the previous territory, move to the next position ($X = X_{next}$) and add n by 1. Keep jumping in the right direction (+) and accelerating ($n = n + 1$) until the next territory is bad. The red X shows the last territory of acceleration. If the acceleration is executed, this is the first situation. The second situation is that no better direction is found in the beginning. Next, the jaguar will decelerate and check other territories nearby in both situations.

3) JUMPING DISTANCE REDUCTION

In the first situation, after acceleration, keep reducing n by 1 ($n = n - 1$) and checking both sides ($X + d \times 2^n$, $X - d \times 2^n$) until both new territories are the same as the current territory. If the found territory is better than the previous one, the green solid line shows the movement. If the found territory is not better than the previous one, the green dotted line shows the movement. The example is shown in Fig. 9 with tendency (+).

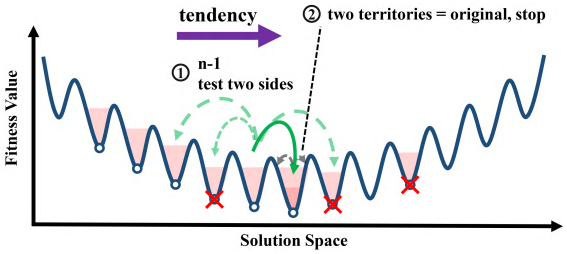

FIGURE 9. Jumping distance reduction in direction (+).

In the second situation, if there is no better direction at the beginning (Adaptive Territory), as previously, keep reducing the hunting range of the first territory and try both sides until the same territory is found. The pseudocode for this process is shown in Fig. 10.

```
JumpingDistanceReduction()
Input: DIST,n,t
Do
  n--;
  find two territories of t with X_R = t.X + DIST * 2^n; X_L = t.X - DIST * 2^n;
  Hunting with X_R and X_L;
  t = Best(t_L, t, t_R);
While t_R and t_L are both new territories
Return t;
```

FIGURE 10. Jumping distance reduction pseudocode.

E. FORBIDDEN TENDENCY

In most cases of the adaptive territory jumping mechanism, the jaguar is able to find the best territory, which contains the best solution in the tendency range. However, many territories have already been claimed in the jumping mechanism. Each territory contains its range. If the territories found in the jumping mechanism are connected, the tendency can be obtained. The lower bound of the tendency is the position (X) of the leftmost territory minus its hunting range (r), while the upper bound of the tendency is the position (X) of the rightmost territory plus its hunting range (r). In other words, it is able to determine whether the tendency range has been searched. Moreover, with the idea of territoriality, the jaguar is able to forbid the tendency because the best territory in this tendency has been found. Other jaguars will therefore not enter this tendency. The example is shown in Fig. 11.

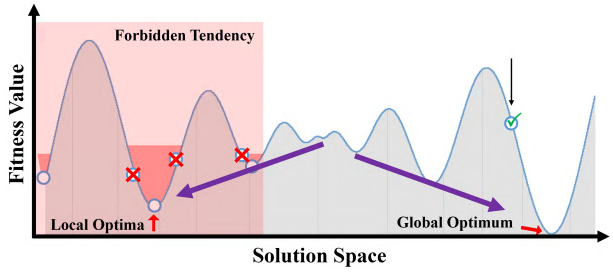

FIGURE 11. Forbidden tendency in function with two tendencies.

When the jaguar is initially in an unknown area, it continues to hunt, claim territory and jump. As a result, the jaguar has a better chance of finding the best solution in other tendencies. After this, previous tendency will be forbidden. At the same time, if the new tendency shares the same best solution with the previous tendency, the two will be merged. In summary, the tendency range is expandable and searching efficiency will be enhanced.

F. LEARNING

The learning method is designed by mimicking the learning behavior of jaguars. In nature, cubs follow their mother and learn how to hunt. In JA, while the mother moves in this dimension, cubs hunt in other dimensions. Learning is similar to the adaptive territory jump mechanism. The difference is that the jaguar considers one dimension in the adaptive territory jump mechanism, but multiple dimensions in learning when moving because in the functions with dimensionality, it would not be possible to find better solutions when only considering one dimension in jumping. Learning mechanism (Fig. 12) considers all the dimensions at the same time, follows the tendency of territories, and it can solve the dependency problem.

1) DECIDE THE DIRECTION AND DISTANCE (VECTOR)

JA finds the good direction in each dimension. In this paper, JA uses the direction between the initial position and final position in learning mechanism to find the better direction. According to the good direction, jaguars move the distance of a step (step is the only adaptive parameter in JA, it is designed according to IEEE754 [28] and the domain of the problem) in each dimension. The direction and the distance in every dimension compose a vector.

2) ACCELERATION

JA keeps doubling the vector, doing HuntingByTurns (Fig. 13) and updating the vector until the fitness of the position is worse than the current position. The example of expected movement of learning is shown in Fig. 14. JA finds the better position, doubles the vector, and updates the vector based on new better position, then keep searching until finding worse one.

3) DECELERATION

In deceleration, JA keeps trying two sides where may exist the better solution. JA reduces the vector by half, does HuntingByTurns and updates the vectors until the positions of the two sides are equal to the current position.

IV. EXPERIMENT AND ANALYSIS

This section shows the results of experiments conducted to evaluate the performance of the proposed Jaguar Algorithm. Seven classic benchmark function problems are used to test the capabilities of the compared algorithms. These functions are taken from [12], [13], [26]. In the first subsection, the characteristics and the difficulties of each function are

```
Learning Mechanism
Input: current position X
// Find the direction and distance (vector V)
For d = 0, d < MaxDimension, d++
If the better direction in dimension d is right (+) side
  V_d = initial step;
End if
If the better direction in dimension d is left (-) side
  V_d = - initial step;
End if
End for
// Find the tendency
While not finding new peak and not reaching bound in each side
  If not finding new territory in Positive side and not reaching bound
  X_P = X + V;
  HuntingByTurns(X_P);
  End if
  If not finding new territory in Negative side and not reaching bound
  X_N = X – V;
  HuntingByTurns(X_N);
  End if
  V=V*2;
End while
If f(X_P) or f(X_N) is the best
  X = Best(X_P, X_N); // Move to the better position
  V = Best(X_P, X_N) - X; // Update the vector
  // Acceleration
  do
  X_next = X + V;
  HuntingByTurns(X_next);
  if f(X_next) is better than f(X)
    V = 2 * (X_next – X);
    X = X_next;
  End if
  While new position is better
Positive vector V_P = 0.5 * (X_next - X);
Negative vector V_N = - 0.5 * (X_next - X);
Else
Positive vector V_P = 0.5 * (X_P - X);
Negative vector V_N = 0.5 * (X_N - X);
End if
// Deceleration
do
X_P = X + V_P;
X_N = X + V_N;
  HuntingByTurns(X_P and X_N);
  If f(X_P) is better than f(X) and f(X_N)
    V_P = X_P –X;
    V_N = - (X_P –X);
    X=X_P;
  Else if f(X_N) is better than f(X) and f(X_P)
    V_N = X_N –X;
    V_P = - (X_N –X);
    X=X_N;
Else
    V_N = 0.5 * (X_N –X);
    V_P = 0.5 * (X_P –X);
  End if
While X_P and X_N are not equal to X
```

FIGURE 12. Learning pseudocode.

```
HuntingByTurns()
Input: X={X_0...X_MaxDimension}
For all dimension set step = initial step
While the step is not too small to be added on position X
  For d = 0, d < MaxDimension, d++
    RushAndApproach(X_d,step_d);
  End for
  step_d = step_d / 2.0;
End while
```

FIGURE 13. HuntingByTurns pseudocode.

introduced, and used to test the capability of an algorithm to address different types of problems. In the following subsection, the reason and purpose of each capability of JA is

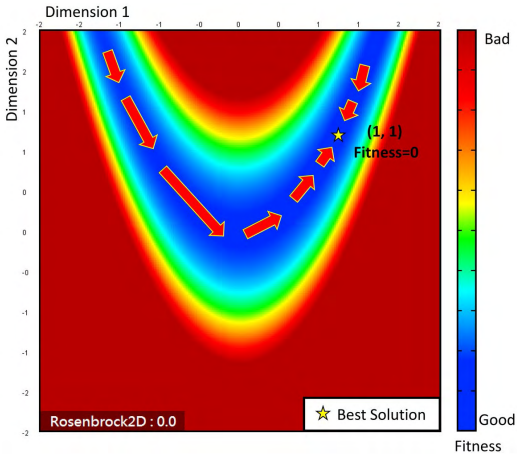

FIGURE 14. Learning follows the tendency (Acceleration).

individually analyzed. Each capability is tested with different functions to examine whether these capabilities take their own role. The order of the self-analysis experiment is the same as for the flow of the Jaguar Algorithm, namely hunting, adaptive initial step, adaptive territory jumping mechanism, forbidden tendency and learning. The results of all the benchmark functions are shown in section IV-B.6 The section IV-C presents comparisons of the JA and other algorithms. The settings of algorithms for the experiment environment are given in section IV-C, including several common control parameters for different experiments, which are population size, the stopping criteria of the algorithms and the dimension of the function. In addition, it also notes the control parameters that other algorithms use to balance exploration and exploitation. Different function optimization problems are used to test the performance of the algorithms. JA is implemented in Visual Studio 2012 C++. The test environment is set up on a personal computer with a Pentium E5300 2.6 GHz CPU, 4G RAM, and running Windows 7.

A. BENCHMARK FUNCTIONS

Various types of functions are used to test different capabilities of algorithms. For example, unimodal problems only have one peak. Algorithms with good exploitation can find the global optimum efficiently. Multimodal functions have two or more local optima. It is easy to make algorithm trapped in local optima. The difficulty level of each function will increase if the global optimum is far from the second or third optimum, and so on. The algorithms are easily attracted to local optima and become stuck in them. In addition, the functions with dependency are worth discussing. The initial research of dependency was published in 1994 by Friedman [29]. The important point here is that variables of dependency problems interact with each other. Sometimes, these problems must consider more than one variable in order to obtain a better solution. The following will simply introduce the characteristics of benchmark functions one by one. In next subsection, JA will be implemented on these

TABLE 1. Benchmark function list.

Function Name	Function	Global Optima	Domain
Absolute Value	$f(x)=\sum_{i=1}^{d}\lvert x_i\rvert$	0 at x=(0,0,...,0)	[-100,100]
Sphere	$f(x)=\sum_{i=1}^{d}x_i^2$	0 at x=(0,0,...,0)	[-100,100]
Rastrigin	$f(x)=10d+\sum_{i=1}^{d}(x_i^2-10cos(2\pi x_i))$	0 at x=(0,0,...,0)	[-5.12,5.12] [-15,15]
Ackley	$f(x)=-20e^{-0.2\sqrt{\frac{1}{d}\sum_{i=1}^{d}x_i^2}}-e^{\frac{1}{d}\sum_{i=1}^{d}cos(2\pi x_i)}+20+e$	0 at x=(0,0,...,0)	[-32,32] [-32.768,32.768]
Schwefel	$f(x)=418.9829d-\sum_{i=1}^{d}x_isin(\sqrt{x_i})$	0 at x=(420.$\overline{9687}$,420.$\overline{9687}$,...,420.$\overline{9687}$)	[-500,500]
Griewank	$f(x)=1+\frac{1}{4000}\sum_{i=1}^{d}x_i^2-\prod_{i=1}^{d}cos(\frac{x_i}{\sqrt{i}})$	0 at x=(0,0,...0)	[-600,600]
Rosenbrock	$f(x)=\sum_{i=1}^{d-1}[100(x_i^2-x_{i+1})^2+(1-x_i)^2]$	0 at x=(1,1,...,1)	[-2.048,2.048] [-15,15] [-30,30]

complicated problems, and self-analysis will be conducted. JA will then be compared to other algorithms.

The first function is Absolute Value, whose global minimum is at x = 0, 0, , 0, and its value is 0. The domain for the function is [−100, 100]. There is only a valley in this function, and its global optimum is located at the center of the valley. This kind of function without any local optimum is called a unimodal function. It is applied to test the exploitative ability of an algorithm.

Sphere is also a unimodal function whose global minimum is at x = 0, 0, , 0, and its fitness is 0. The domain for the function is [−100, 100]. It is also test the force of exploitation of an algorithm.

The Rastrigin function is transformed from the Sphere function. The global minimum is at x = 0, 0, , 0 and its value is 0. The common domain for the function is [−15, 15]. This function is a multimodal function. Due to the addition of cosine modulation, the function has many local optima. As a result, an algorithm cannot just have the exploitative ability; otherwise it would be trapped in the local optima. Algorithms must have the explorative ability and can escape from the local optima.

The Ackley function is a multimodal function whose global minimum is at x = 0, 0, , 0, and its value is 0. The common domain for the function is [−32.768, 32.768]. There are numerous local optima owing to an exponential term. In order to achieve the global optimum, both exploration and exploitation are essential.

The global minimum of Schwefel function is at x = 420.9687, 420.9687, …, 420.9687. The domain of the function is [−500, 500]. It is worth noting that there are two tendencies in each dimension; however, the global optimum is only located on one side, and the global optimum is far from the second-best optimum. If the dimension is 30, there are 2^{30} tendencies, and the global optimum exists in only one of these tendencies; thus, an algorithm would be easily trapped in the local optima.

The global minimum of the Griewank function is at x = 0, 0, , 0, and its value is 0. The domain for the function is [−600,600]. The Griewank function has a product term that makes variables have slight dependency. Thus, although it is a multimodal function, the problem appears to be unimodal when the dimension is higher than 30.

The last function is the Rosenbrock function, whose global minimum is at x = 1, 1, , 1, and its value is 0. The common domain for the function is [−2.048, 2.048]. The global optimum is inside a long, narrow, parabolic-shaped flat valley. The Rosenbrock function has a chain-like interaction between the variables. This means that algorithms cannot only consider one variable; they must consider all the variables at the same time in order to find the optimal solution. It is a strong dependency problem.

B. SELF-ANALYSIS

This section provides an analysis of the capabilities of the proposed method. Each function has different characteristics. Experiments and analysis for different functions are presented. The following experiments will discuss the capabilities of JA in detail. After this, JA will be applied to the benchmark functions listed in Table 1.

1) HUNTING

This section tests exploitation capabilities on unimodal function problems, e.g. the absolute function and the sphere function. Exploitation in JA is hunting, and it applies the adaptive *initial step* in order to hunt with an appropriate parameter. Hunting means checking both sides of the current position with the *step* to determine whether there is a better adjacent solution. The step is continuously reduced by half until it is too small for the current position to be distinguished from either side. JA uses the concept of the bisectional squeeze theorem to approximate to the possible best position. So, hunting can find the best position in the current area. The result in 30 dimensions is shown in Table 2, and JA compares with other algorithms in Table 3 and 4. JA can find the global optimal solution for the unimodal function problems in 30 trials. In addition, the concept of the bisection method allows JA to exploit more rapidly. JA does not require a large number of evaluations to find the best solution of the current peak, and compared with other algorithms, JA can find the optimal solution faster and more efficiently.

TABLE 2. Result of hunting.

Algorithm	JA	
Function	Average Best Fitness	Evaluation
Absolute[±100]	0.0 (100%)	9,662
Sphere[±100]	0.0 (100%)	9,662

TABLE 3. Compare with other algorithms in absolute.

	Algorithm	PSO	IMQTS	JA
Absolute [±100]	Fitness	3.53E-01 ±2.87E+00	9.32E+00 ±1.3E+00	0E+00 ±0E+00
	Evaluation	30,000	750,000	9,662

TABLE 4. Compare with other algorithms in sphere.

	Algorithm	QEA	FEP	JA
Sphere [±100]	Fitness	1.8E-04 ±1.3E-04	5.7E-04 ±1.3E-04	0E+00 ±0E+00
	Evaluation	150,000	150,000	9,662

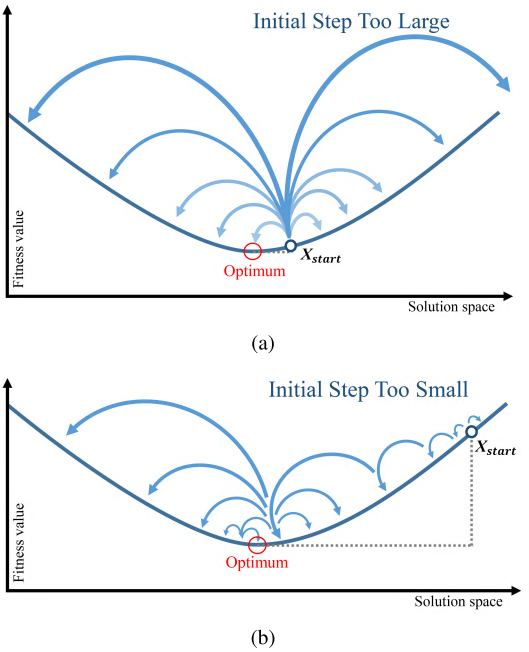

FIGURE 15. Instances of the initial step which is too large or too small. (a) The initial step is too large. (b) The initial step is too small.

2) ADAPTIVE INITIAL STEP

In the hunting process, the jaguar tries both sides of its current position with positive and negative directions in order to identify the best direction. In this step, there is a parameter which must be set, namely *initial step*. This section explains how the *initial step* is formulated, and describes two methods of producing the *initial step*. Initial steps that are too large or small are not appropriate for hunting. Large steps have to be divided many times to find optimal solutions in many cases(as in Fig. 15(a)). On the other hands, small steps require much more time in many cases (as in Fig. 15(b)). JA adaptively adjusts the best *initial step* when facing different problems. Hence, the series of fixed initial steps are tested on five different functions, from feasible maximal step to minimal step. The step can be applied to every position within the domain. The experiment then notes how many evaluations were required for the hunting method using each initial step (i.e. the speed of finding the optimal solution in its located region). The test experiment is run 100,000 times, and the blue line in Fig. 16 shows the evaluations.

Using the above testing, this study proposes two ways of producing the initial step. The first uses the domain range, which is defined in the beginning. The second uses the value of the initial position, which is randomly initialized in the beginning. Both values are known at the beginning of the experiments. In Fig. 16, the middle size of initial steps cost a relatively small amount of evaluation. Therefore, the adaptive initial step of JA is formulated by the middle value of maximal and minimal precision, generated by both methods.

The results of the adaptive initial step experiment on different function problems are given in Fig. 16, where the red line represents the domain and the green line represents the initial position. Both adaptive methods cost fewer evaluations to rapidly hunt the optimum of an area, and the same is true for the other functions. These two methods are thus able to adaptively produce the initial steps. These two methods have their own features. They both result in more rapid hunting, with lower evaluation costs, resulting in efficient exploitation.

3) ADAPTIVE TERRITORY JUMPING MECHANISM

In order to test the capability of the adaptive territory jumping mechanism, two kinds of JA with different abilities are compared: (1) hunting without jumping; (2) hunting with jumping. Each method is executed in 30 independent experiments. The result is shown in Table 5. The test function is the Rastrigin function of one dimension in Table 5. The solution space contains many local optima and one global optimum located in the center. This function has a tendency for the local optima to become better as they get closer to the center. It is thus very easy to become trapped in the local optimum. The jaguar has the ability to find the global optimum with the adaptive territory jumping mechanism.

TABLE 5. Result of adaptive territory jumping mechanism.

Rastrigin[±15] 1 Dimension	Average Best Fitness	Evaluation
JA without Jumping	0.232 (73%)	586
JA with Jumping	0 (100%)	539

4) FORBIDDEN TENDENCY

In territoriality, after jumping, the jaguar is able to leave tendencies that have been searched. In Table 6, Hunting, the jumping mechanism and the forbidden tendency are implemented. The result is obtained by 30 independent experiments.

The Schwefel function is used in 30 dimensions. There are two opposite tendencies in each dimension. The best solution

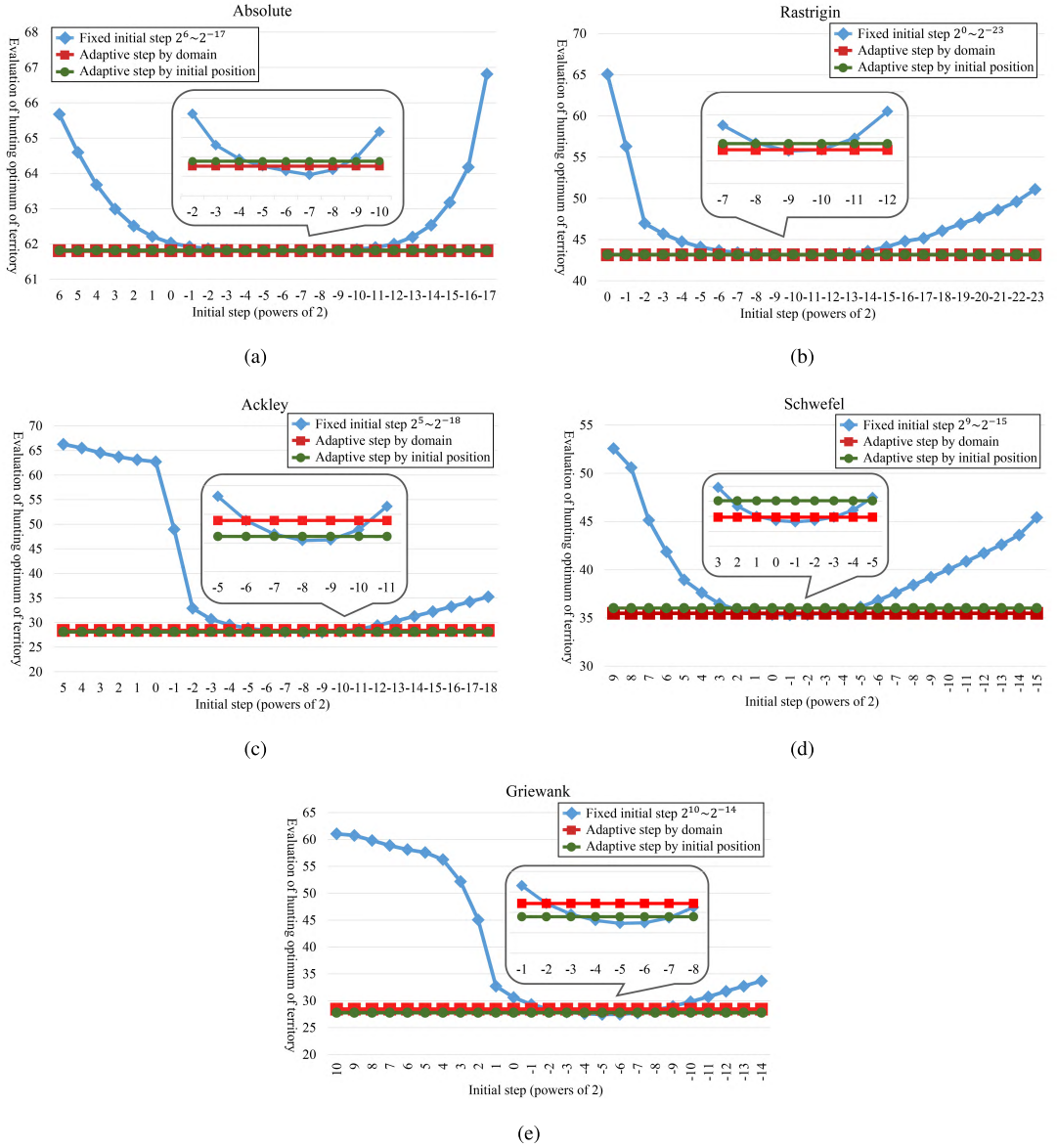

FIGURE 16. Comparison of every feasible initial step and two adaptive initial step methods in five functions. (a) Absolute. (b) Rastrigin. (c) Ackley. (d) Schwefel. (e) Griewank.

TABLE 6. Result of forbidden tendency.

Schwefel[±500] 30 Dimensions	Average Best Fitness	Evaluation
JA with Forbidden tendency	0.00382±0 (100%)	39,559

is in the right/positive tendency. It contains $2^{dimension}$ tendencies in the whole solution space and each tendency consists of many peaks. In 30 dimensions, there are 2^{30} (1,073,741,824) tendencies, and the optimal solution is only in one tendency. In that one tendency, the optimal solution is located near the bound. Because the structure is very complicated, traditional algorithms are very easily attracted to converge into local optima. However, in the proposed method, a tendency consists of many territories. The forbidden tendency mechanism is the same idea as territory mechanism. JA will therefore not waste evaluations on tendencies previously found. This makes the JA search solution space more efficient, and less likely to be trapped in local optima. As a result, JA finds the best solution with fewer evaluations in each experiment.

5) LEARNING

In order to address the dimensionality of functions, JA implements the above abilities and learning in this experiment. Tables 7 and 8 give the average results of 30 independent experiments. In Table 7, the success rate refers to the percentage that the optimal solution is successfully found in 30 independent experiments. In Table 8, the values of the first column d represent the function dimensions.

TABLE 7. Comparison result: JA without learning vs. JA with earning.

Griewank[600] 2 Dimensions	Without Learning	With Learning
Successful Rate	60%	100%

TABLE 8. Result of learning in 2 and 30 dimension functions.

d	Function	Average Best Fitness	Evaluation
2	Griewank[±600]	0±0	10,618
	Rosenbrock[±2.048]	0±0	17,767
30	Griewank[±600]	0±0	263,311
	Rosenbrock[±2.048]	0±0	199,462

The benchmark function is the Rosenbrock function in 2 and 30 dimensions. It is known that in the Rosenbrock function, every dimension is connected to the others; the dimensionality is very strong. The Griewank function is also a dependency problem. Most algorithms do not perform well with these functions; the results are shown in the next subsection. However, the jaguar is able to move in multiple dimensions at the same time. Therefore, JA has better ability to find optimal solution and yields a good result in dependency problem.

TABLE 9. Result of JA tested on 30 dimension functions.

Functions 30 Dimensions	Average Best Fitness	Evaluation
Absolute Value[±100]	0±0	29,218
Sphere[±100]	0±0	29,221
Rastrigin[±5.12]	0±0	36,818
Rastrigin[±15]	0±0	47,219
Ackley[±32]	0±0	66,697
Ackley[±32.768]	0±0	69,428
Schwefel[±500]	0.000382±0	39,559
Griewank[±600]	0±0	263,311
Rosenbrock[±2.048]	0±0	199,462

6) NUMERICAL OPTIMIZATION

In this section, JA implements all the behaviors in the experiment, as shown in Table 9. JA is tested with the following functions in 30 dimensions: Absolute Value, Sphere, Rastrigin, Ackley, Griewank, Schwefel and Rosenbrock. JA has many features that help the jaguar find the global optimum in these functions. JA is able to find the best solution in an area rapidly using an adaptive initial step and hunting. Therefore, the jaguar is able to find the optimal solution in the Absolute Value and Sphere unimodal functions. The jaguar has the ability to find the best territory in a tendency by using the adaptive territory jumping mechanism. This helps the jaguar find the optimal solution in multi-modal functions Rastrigin and Ackley. Moreover, Schwefel is a function with multi-tendencies. The jaguar is able to leave from a tendency that has already been searched. This allows JA has more opportunities to find the global optimum. In the Griewank and Rosenbrock functions with dependency, the jaguar is able to move multiple dimensions at the same time through learning. As a result, in the experiments, JA has great ability to find the optimal solution in these functions.

TABLE 10. Result of JA comparison with QEA and FEP.

Function	QEA	FEP	JA
Sphere [±100]	1.8E-04 ±1.3E-04	5.7E-04 ±1.3E-04	**0E+00 ±0E+00**
	150,000	150,000	**29,221**
Ackley [±32]	2.5E-03 ±8.1E-04	1.8E-02 ±2.1E-03	**0E+00 ±0E+00**
	150,000	150,000	**66,697**
Griewank [±600]	3.6E-02 ±3.2E-02	1.6E-02 ±2.2E-02	**0E+00 ±0E+00**
	200,000	200,000	263,331
Rastrigin [±5.12]	3.9E-02 ±1.9E-01	4.6E-02 ±1.2E-02	**0E+00 ±0E+00**
	500,000	500,000	**36,818**
Schwefel [±500]	3.8E-04 ±3.0E-09	1.4987E+01 ±5.26E+01	**3.82E-04 ±0E+00**
	900,000	900,000	**39,559**
Rosenbrock [±30]	1.173E+01 ±1.836E+01	5.06E+00 ±5.87E+00	**0E+00 ±0E+00**
	2,000,000	2,000,000	**251,218**

TABLE 11. Result of JA comparison with PSO and IMQTS.

Function	PSO	IMQTS	JA
Absolute [±100]	3.53E-01 ±2.87E+00	9.32E+00 ±1.3E+00	**0E+00 ±0E+00**
	>30,000	750,000	**29,218**
Ackley [±32.768]	2.49E+00 ±1.35E+00	2.7259E+00 ±3.538E-01	**0E+00 ±0E+00**
	>30,000	200,000	**69,427**
Griewank [±600]	3.72E-02 ±5.26E-02	4.02E-02 ±1.31E-02	**0E+00 ±0E+00**
	>30,000	2,500,000	**263,311**
Rastrigin [±5.12]	6.66E+01 ±1.71E+01	4.10037E+01 ±3.022E+01	**0E+00 ±0E+00**
	>30,000	1,500,000	**36,818**
Rosenbrock [±2.048]	2.65E+01 ±1.53E+01	1.136256E+02 ±1.0676E+02	**0E+00 ±0E+00**
	>30,000	100,000	199,462

C. COMPARISON

In this section, JAs performance is compared with traditional algorithms with different functions. JA uses one jaguar in the following experiments. The terminal criterion of JA is shown in Fig. 1, which breaks the traditional meta-heuristic framework. The adaptive initial step is produced by domain range. In traditional algorithms, the terminal condition is the maximum number of generations. However, the most important criterion should be the number of evaluations. In one generation, the algorithms are able to implement a huge number of populations and local searches. However, it seems unfair to simply compare JA with other algorithms with different populations and generations. The evaluation times of traditional algorithms are estimated by the number of populations and generations. The reasonable comparison between algorithms would be to examine the number of evaluations required for an algorithm to find a better solution. The results in this section are shown in Tables 10 to 12. Each cell consists of two rows and one column. The values in the upper row represent the average of best solutions in 30 independent experiments and standard deviation. The value in the lower row represents the evaluation cost.

TABLE 12. Comparisons of JA with GA, PSO, PS-EA and ABC.

Function	GA	PSO	PS-EA	ABC	JA
Ackley [±32.768]	1.0989E+00 ±2.4956E-01	1.4917E-06 ±1.8612E-06	3.771E-01 ±9.8762E-02	0.0E+00 ±0.0E+00	**0E+00 ±0E+00**
	125,000	125,000	125,000	250,000	**69,428**
Griewank [±600]	1.2342E+00 ±1.1045E-01	1.1151E-02 ±1.421E-02	8.211E-01 ±1.394E-01	0.0E+00 ±0.0E+00	**0E+00 ±0E+00**
	125,000	125,000	125,000	250,000	263,311
Rastrigin [±15]	1.04388E+01 ±2.6386E+00	3.2476E+01 ±6.9521E+00	3.0527E+00 ±9.985E-01	0.0E+00 ±0.0E+00	**0E+00 ±0E+00**
	125,000	125,000	125,000	250,000	**47,220**
Schwefel [±500]	1.35346E+01 ±4.9534E+00	9.9077E+02 ±5.8114E+02	3.272E+00 ±1.6185E+00	3.82E-04 ±1.0E-12	**3.82E-04 ±0E+00**
	125,000	125,000	125,000	250,000	**39,559**
Rosenbrock [±15]	1.66283E+02 ±5.95102E+01	4.0254E+02 ±6.3365E+02	9.8407E+01 ±3.55791E+01	2.0121E-02 ±2.1846E-02	**0E+00 ±0E+00**
	125,000	125,000	125,000	250,000	180,430

The comparison with QEA [13] and FEP [13] is shown in Table 10. The common test parameter settings of the experiment are as follows: 50 independent experiments, 30 dimensions and 100 populations. The terminal conditions for QEA and FEP differ for each function (Sphere 1500, Ackley 1500, Griewank 2000, Rastrigin 5000, Schwefel 9000, Rosenbrock 20000). The exclusive control parameters of QEA given in [13] are for H gate $= 0.01$, and the encode length of Q-bits employed for functions (Sphere 18, Ackley 18, Rosenbrock 18, Griewank 21, Rastrigin 17, Schwefel 22). The control parameters of FEP given in [30] are t $= 1$ for Cauchy mutation and q $= 10$ for selection.

In the Sphere, Ackley, Griewank, and Rastrigin functions, JA can find the optimal solution with a fewer number of evaluations compared with QEA and FEP. In Schwefel function, JA also finds the optimal solution. Compare with QEA, JA outputs different precision, it does not mean that JA finds a worse solution than QEA. In the Rosenbrock function, although JA uses more evaluations, it still finds a better solution than those found by QEA and FEP.

The algorithms presented were from [10] and [26] in this comparison experiment, with 30 dimension functions. The common test parameter settings for the experiment are as follows. There are 50 independent experiments. The population size and termination criteria (maximal generation) are different. There are 30 populations and 1000 generations in PSO. The populations and generations used by IMQTS for each function are 1500 and 500 for Absolute Value, 1000 and 200 for Ackley, 5000 and 500 for Griewank, 1500 and 1000 for Rastrigin and 500 and 200 for Rosenbrock. Owing to the different numbers of populations and generations, it is more reasonable to use the number of evaluation as comparison criteria. Therefore, the number of evaluations is calculated for each algorithm and its varying population and generations, as shown in Table 11. The exclusive control parameters of PSO given in [5] are inertia weight w $= 0.729844$, learning factors $c1, c2 = 1.496180$ and initial velocity $v(t = 0) = 0$. The control parameters of IMQTS given in [15] are (a, b, c) for a bell-shaped function which is employed with the algorithm. Respectively, (a, b, c) are (300, 400, 1) for Absolute Value, (200, 300, 300) for Ackley, (100, 150, 100) for Griewank, (600, 700, 1300) for Rastrigin and (100, 10, 40) for Rosenbrock.

In Table 11, the result shows that JA can find the optimal solution in all five functions. Compared with IMQTS, JA uses less number of evaluations to obtain a better solution for most functions.

The algorithms presented were from [31], and [12]. JA is compared with these algorithms using 30 dimension functions, with results given in Table 12 The common test parameter settings for the experiment are as follows: 30 independent experiments and 125 populations. The terminal criteria (maximal generation) are 1000 in GA, PSO and PS-EA, and 2000 in ABC. The exclusive control parameters of GA given in [31] are single point crossover rate $= 0.95$ and Gaussian mutation rate $= 0.1$. The control parameters of PSO given in [31] are inertia weight varying from 0.9 to 0.7 linearly with the iterations, and learning factors $c1, c2 = 1.496180$. The control parameters of PS-EA given in [31] are inheritance probabilities which are adjusted dynamically. A feasible set of initial inheritance probabilities is used to test the performance in [31]. The control parameters of ABC given in [12] are a population with 50% employed bees and 50% onlooker bees. There is one scout bee. The limit (i.e. the number of generations after which the food source is abandoned) is set according to population and dimensions.

In Table 12, the result shows that JA can find the optimal solution for all functions. In Rosenbrock, the dimensionality of the function causes most algorithms to perform poorly. If the domain range is larger, the difficulty also increases. Compared with ABC in the Ackley, Rastrigin, Schwefel, and Rosenbrock functions JA uses fewer evaluations to obtain the optimal solution.

These experiment results demonstrate JAs superior performance. During hunting, JA uses fewer evaluations to find the optimal solution than do other algorithms in unimodal functions. With territoriality and jumping, JA uses an effective way to find the optimal solution in multimodal functions.

With forbidden tendencies, JA can find the global optimum in Schwefel, while other algorithms are easily trapped in local optima. With learning, JA can move in multiple dimensions at the same time. JA achieves good performance in functions with dimensionality (dependency). Compared with other traditional algorithms, it shows that learning can be improved in evaluations. As a result, JA is able to find the optimal solution faster and with less required resources compared to traditional algorithms.

V. CONCLUSION

Traditional optimization algorithms are designed to balance the capabilities of exploration and exploitation in order to obtain an optimal solution with limited costs or time. In order to effectively balance exploration and exploitation, many parameters are implemented, such as weights and random models. Algorithms then need to adjust these parameters for different problems, and random models waste computational cost and make these algorithms very difficult to analyze. Different from traditional methods, JA performs exploration and exploitation separately. It can, therefore, engage in exploration and exploitation at the right moment for each.

JA consists of five features: hunting, territoriality, an adaptive territory jumping mechanism, forbidden tendency, and learning. JA is simple and is implemented without randomness. Moreover, the only parameter, the step, is adaptive to the environment. The mechanisms of hunting, jumping, and learning share the same concept. In hunting, a jaguar follows the best direction and accelerates. If JA meets a bad position, it decelerates and checks both sides of the current position. Using the hunting process, the jaguar is able to find the best solution in an area within a short time and small cost. With the adaptive step, the jaguar is able to hunt more rapidly. Territoriality prevents other jaguars or itself from entering regions that have already been searched; this also allows the jaguar to search the solution space more efficiently, without wasting extra evaluations on these regions and prevent it trapping in local optima. When new territory is claimed, the jaguar checks the appropriate range of the territory and uses this information to jump. With the adaptive territory jumping mechanism, the jaguar can follow the best tendency of territories. After this, the jaguar leaves the tendency, making JA more efficient. With the learning mechanism, the jaguar is able to move in multiple dimensions at the same time, making JA more efficient, and gives it the ability to deal with dependency problems. JA differs from the strategy of balancing exploration and exploitation in traditional algorithms, as JA implements exploration and exploitation separately. The results of the benchmark tests show that JA outperforms the traditional algorithms, and have powerful search ability to find the global optimal solution in a much lesser computational cost.

REFERENCES

[1] B.-Y. Liao, H.-W. Chen, S.-Y. Kuo, and Y.-H. Chou, "Portfolio optimization based on novel risk assessment strategy with genetic algorithm," in *Proc. IEEE Int. Conf. Syst., Man, Cybern.*, Oct. 2015, pp. 2861–2866.

[2] Y.-H. Chou, S.-Y. Kuo, C.-Y. Chen, and H.-C. Chao, "A rule-based dynamic decision-making stock trading system based on quantum-inspired tabu search algorithm," *IEEE Access*, vol. 2, pp. 883–896, 2014.

[3] S.-Y. Kuo, C. Kuo, and Y.-H. Chou, "Dynamic stock trading system based on quantum-inspired tabu search algorithm," in *Proc. IEEE Congr. Evol. Comput.*, Jun. 2013, pp. 1029–1036.

[4] S.-Y. Kuo, Y.-H. Chou, and C.-Y. Chen, "Quantum-inspired algorithm for cyber-physical visual surveillance deployment systems," *Comput. Netw.*, vol. 117, pp. 5–18, Apr. 2017.

[5] J. G. Digalakis and K. G. Margaritis, "An experimental study of benchmarking functions for genetic algorithms," *Int. J. Comput. Math.*, vol. 79, no. 4, pp. 403–416, 2002.

[6] R. Storn and K. Price, "Minimizing the real functions of the ICEC'96 contest by differential evolution," in *Proc. IEEE Int. Conf. Evol. Comput.*, May 1996, pp. 842–844.

[7] S. Das and P. N. Suganthan, "Differential evolution: A survey of the state-of-the-art," *IEEE Trans. Evol. Comput.*, vol. 15, no. 1, pp. 4–31, Feb. 2011.

[8] J. Knowles and D. Corne, "Approximating the nondominated front using the pareto archived evolution strategy," *Evol. Comput.*, vol. 8, no. 2, pp. 149–172, Jun. 2000.

[9] J. R. Koza, *Genetic Programming—On the Programming of Computers by Means of Natural Selection*, vol. 1. Cambridge, MA, USA: MIT Press, 1992.

[10] A. Engelbrecht, "Particle swarm optimization: Velocity initialization," in *Proc. IEEE Congr. Evol. Comput. (CEC)*, Jun. 2012, pp. 1–8.

[11] M. Dorigo, V. Maniezzo, and A. Colorni, "Ant system: Optimization by a colony of cooperating agents," *IEEE Trans. Syst., Man, Cybern. B, Cybern.*, vol. 26, no. 1, pp. 29–41, Feb. 1996.

[12] D. Karaboga and B. Basturk, "A powerful and efficient algorithm for numerical function optimization: Artificial bee colony (ABC) algorithm," *J. Global Optim.*, vol. 39, no. 3, pp. 459–471, 2007.

[13] K.-H. Han and J.-H. Kim, "Quantum-inspired evolutionary algorithms with a new termination criterion, H_ϵ gate, and two-phase scheme," *IEEE Trans. Evol. Comput.*, vol. 8, no. 2, pp. 156–169, Apr. 2004.

[14] Y.-H. Chou, C.-Y. Chen, C.-H. Chiu, and H.-C. Chao, "Classical and quantum-inspired electromagnetism-like mechanism and its applications," *Control Theory Appl., IET*, vol. 6, no. 10, pp. 1424–1433, Jul. 2012.

[15] Y.-H. Chou, C.-H. Chiu, and Y.-J. Yang, "Quantum-inspired tabu search algorithm for solving 0/1 knapsack problems," in *Proc. 13th Annu. Conf. Companion Genet. Evol. Comput.*, 2011, pp. 55–56.

[16] H.-P. Chiang, Y.-H. Chou, C.-H. Chiu, S.-Y. Kuo, and Y.-M. Huang, "A quantum-inspired tabu search algorithm for solving combinatorial optimization problems," *Soft Comput.*, vol. 18, no. 9, pp. 1771–1781, 2014.

[17] F. Glover, "Tabu search—Part I," *ORSA J. Comput.*, vol. 1, no. 3, pp. 190–206, 1989.

[18] Q. Qin, S. Cheng, Q. Zhang, L. Li, and Y. Shi, "Particle swarm optimization with interswarm interactive learning strategy," *IEEE Trans. Cybern.*, vol. 46, no. 10, pp. 2238–2251, Oct. 2016.

[19] M. R. Bonyadi and Z. Michalewicz, "Stability analysis of the particle swarm optimization without stagnation assumption," *IEEE Trans. Evol. Comput.*, vol. 20, no. 5, pp. 814–819, Oct. 2016.

[20] J. Liu, Y. Mei, and X. Li, "An analysis of the inertia weight parameter for binary particle swarm optimization," *IEEE Trans. Evol. Comput.*, vol. 20, no. 5, pp. 666–681, Oct. 2016.

[21] G. Zhu and S. Kwong, "Gbest-guided artificial bee colony algorithm for numerical function optimization, " *Appl. Math. Comput.*, vol. 217, no. 7, pp. 3166–3173, 2010.

[22] M. Neshat, G. Sepidnam, M. Sargolzaei, and A. N. Toosi, "Artificial fish swarm algorithm: A survey of the state-of-the-art, hybridization, combinatorial and indicative applications," *Artif. Intell. Rev.*, vol. 42, no. 4, pp. 965–997, 2014.

[23] S. Mirjalili, S. M. Mirjalili, and A. Lewis, "Grey wolf optimizer," *Adv. Eng. Softw.*, vol. 69, pp. 46–61, Mar. 2014.

[24] X.-S. Yang, "A new metaheuristic bat-inspired algorithm," *Nature Inspired Cooperat. Strategies Optim.*, vol. 284, pp. 65–74, 2010. [Online]. Available: https://link.springer.com/chapter/10.1007/978-3-642-12538-6_6

[25] K. N. Krishnanand and D. Ghose, "Glowworm swarm optimisation: A new method for optimising multi-modal functions," *Int. J. Comput. Intell. Stud.*, vol. 1, no. 1, pp. 93–119, 2009.

[26] Y.-J. Yang, S.-Y. Kuo, F.-J. Lin, I.-I. Liu, and Y.-H. Chou, "Improved quantum-inspired tabu search algorithm for solving function optimization problem," in *Proc. IEEE Int. Conf. Syst., Man, Cybern. (SMC)*, Oct. 2013, pp. 823–828.
[27] J. Meng, H. G. Wang, Z. Dong, and K. P. Wong, "Quantum-inspired particle swarm optimization for valve-point economic load dispatch," *IEEE Trans. Power Syst.*, vol. 25, no. 1, pp. 215–222, Feb. 2010.
[28] D. Zuras *et al.*, *IEEE Standard for Floating-Point Arithmetic*, IEEE Standard 754-2008, 2008, pp. 1–70.
[29] J. H. Friedman, "An overview of predictive learning and function approximation," in *From Statistics to Neural Networks*. Berlin, Germany: Springer, 1994.
[30] X. Yao, Y. Liu, and G. Lin, "Evolutionary programming made faster," *IEEE Trans. Evol. Comput.*, vol. 3, no. 2, pp. 82–102, Jul. 1999.
[31] D. Srinivasan and T. H. Seow, "Particle swarm inspired evolutionary algorithm (PS-EA) for multiobjective optimization problems," in *Proc. Congr. Evol. Comput. (CEC)*, vol. 4, Dec. 2003, pp. 2292–2297.

YAO-HSIN CHOU received the Ph.D. degree from the Department of Electrical Engineering, National Taiwan University, Taipei, Taiwan, in 2009. He is currently an Associate Professor with the Department of Computer Science and Information Engineering, National Chi Nan University, Puli, Taiwan. He has authored over 60 papers in journals and conference proceedings. His current research interests include computational intelligence, evolutionary computation, artificial intelligence, financial technology, and quantum information science.

SHU-YU KUO received the B.S. and M.S. degrees in computer science and information engineering from National Chi Nan University, Taiwan, in 2012 and 2013, respectively, where she is currently pursuing the Ph.D. degree with the Department of Computer Science and Information Engineering. Her research interests include evolutionary computation, automated trading systems, and wireless sensor network.

LI-SHENG YANG is currently pursuing the master's degree with the Department of Computer Science and Information Engineering, National Chi Nan University. His research interests include evolutionary computation and metaheuristic algorithm.

CHIA-YUN YANG is currently pursuing the master's degree with the Department of Computer Science and Information Engineering, National Chi Nan University. Her research interests include evolutionary computation and visualization tool.