A Generic Form of Evolutionary Algorithms and Manifold Drift Concept

Chidchanok Lursinsap

Department of Mathematics and Computer Science, Chulalongkorn University, Thailand
Email: lchidcha@gmail.com

***Abstract*—Most of optimization problems in various fields are in NP-class. This implies that the time to find the optimum solution of any problem is obviously non-polynomial. Although the development of high speed computer architectures and the concept parallel computing is practically successful, some of these problems are constrained by the problem of tight data dependency which prevents the possibility of deploying a parallel architecture as well as processing to solve the problem. Evolutionary algorithms which are based on *guessing* solutions have been developed to find an acceptable solution in a short time. However, the processing time based on *guessing* to achieve the acceptable solution is unpredictable and uncontrollable. In this paper, we compare the guessing process in some popular algorithms to define a generic structure of searching process and solution finding process. This structure will help develop a new evolutionary algorithm. Furthermore, a new concept of *manifold drift* for avoiding the guessing process in order to speed up the solution search is also discussed.**

***Index Terms*—Optimization, Evolutionary algorithms, opposite gradient search**

I. Introduction

Optimization is a process to find the best solution for a studied problem. There are several academic, business, and social areas such as science, engineering, management requiring the best solutions for the studied problems. For example, one of the problems in this domain is finding the best solution to pack boxes of different sizes as many as possible into the trunk of a truck where the volume is fixed. These problems usually belong to a class of algorithmic problem called *NP class* where the time complexity to find the best solution is non-polynomial, usually exponential. This time complexity is defined by the computational model of *deterministic Turing machine* (DTM). However, it is possible to speed up the best-solution finding process by employing the computational model of *non-deterministic Turing machine* (NDTM). This type of machine has an additional computing module attached to the DTM to perform the guessing process of the next state of NDTM. By guessing, it is possible to find the best solution at the first iteration of the algorithm or in a polynomial time. Unfortunately, the actual random guessing process cannot to be implemented by using the current computer technology because all hardware components are built on the concept of Boolean algebra. To alleviate this obstacle, several evolutionary algorithms [12], [13] have been invented to imitate the random guessing process to some certain limits.

In this paper, we summarize the popular evolutionary algorithms and derive a generic form of those evolutionary algorithms so that any one can further develop more evolutionary algorithms from this generic form with the relevant interpretation on the behaviours of new animal species. Further, we discuss a new approach of very fast solution searching algorithm without any imitation of animal behaviour.

The rest of this paper is organized as follows. Section II summarizes all popular evolutionary algorithms. Section III discusses the generic form of evolutionary algorithm. Section IV introduces a new approach to find the best solution based on manifold searching concept. Section V concludes the paper.

II. Popular Evolutionary Algorithms

Several evolutionary algorithms [3] have been proposed since the classical inventions of simulated annealing and genetic algorithms. Most recently proposed algorithms are based on the observation of the behaviour of one animal or a group of animals such as bee, ant, whale, bat, and others. Only two algorithms, i.e. brain storm optimization and election campaign optimization are based on human activities. All these algorithms are briefly summarized in the following sections.

A. Whale Optimization Algorithm

This method [17] observes the process of chasing a swarm of prey. When a whale finds its prey, it blows bubble to its prey and encircle the prey. However, the whale must search the location of prey which is unknown in advance. Thus, the location of prey is used to represent the solution of the problem. Let $\mathbf{x}_i(t)$ be solution i and $\mathbf{x}_{best}(t)$ be the best solution found so far at time t. Each solution $\mathbf{x}_i(t)$ is evaluated by a fitness function $f(\mathbf{x}_i(t))$. A new solution can be computed from the current solution by the following equations.

$$\begin{aligned}\mathbf{D} &= |C \times \mathbf{x}_{best}(t) - \mathbf{x}_i(t)| && (1)\\ \mathbf{x}_i(t+1) &= \mathbf{x}_{best}(t) - A \times \mathbf{D} && (2)\\ A &= a \times r_1 - a && (3)\\ C &= 2 \times r_2 && (4)\end{aligned}$$

t is the current iteration. $\mathbf{x}_i(t)$ is the i^{th} solution at iteration t. $\mathbf{x}_{best}(t)$ is the best solution found at iteration t. A and C are scalar coefficients. r_1 and r_2 are random numbers. a is a constant linearly decreased from 2 to 0.

Algorithm: Whale Optimization

```
1.  Set iteration count $t = 1$.
2.  Generate a set of solutions $\mathbf{x}_i(t)$, $1 \leq i \leq N$.
3.  For each solution $\mathbf{x}_i(t)$ do
4.      Compute the value of fitness function $f(\mathbf{x}_i(t))$.
5.  EndFor
6.  Let $\mathbf{x}_{best}(t)$ be the solution found so far from step 3.
7.  Repeat
8.  For each decreased value of $a$ from 2 to 0 do
9.      For $i$ from 1 to $N$ do
10.         Compute constants $A$ and $C$ from Eqs. (3)
            and (4).
11.         Set $p$ to a random number in between 0 and 1.
12.         If $p > 0.5$ then
13.           Update $\mathbf{x}_i(t+1)$ by Eq. (2)
14.         else
15.           If $|A| \geq 0.5$ then
16.             Update solution $\mathbf{x}_i(t+1)$ by Eq. (2).
17.           else
18.             Compute the difference between $\mathbf{x}_{best}$
                and $\mathbf{x}_i(t)$.
19.           EndIf
20.         EndIf
21.     EndFor
22. EndFor
23. $t = t + 1$.
24. Until $t > Max$.
```

B. Artificial Bee Colony

This method [5] is based on the process of a bee while collecting the nectar. The location of each nectar source represents a solution of optimization problem. A bee evaluates the information related to the amount of nectar at the current source with respect to the information from all other sources to determine whether it should fly away to other sources. Let $\mathbf{x}_i = [x_{i,j}, x_{i,2}, \ldots, x_{i,d}]^T$ be the i^{th} solution in the swarm. d is the number of dimensions. Solution $\mathbf{x}_i$ generates a new candidate solution $\mathbf{v} = [v_{i,1}, v_{i,2}, \ldots, v_{i,D}]^T$ in the neighborhood of its present position by using the following equations. The first equation computes the probability to collect more nectar at current source determined from the fitness function $fit(\mathbf{x}_i)$ of solution $\mathbf{x}_i$. The second equation computes the new source location apart from source $\mathbf{x}_i$.

$$p_i = \frac{fit(\mathbf{x}_i)}{\sum_{j=1}^{SN} fit(\mathbf{x}_j)} \quad (5)$$

$$v_{i,j} = x_{i,j} + \phi_{i,j}(x_{i,j} - x_{k,j}) \quad (6)$$

where $-1 \leq \phi_{i,j} \leq 1$ is a random constant. If there are too many close nectar sources, then instead of using $\mathbf{v}_i$, $x_{i,j}$ is redefined as follows.

$$x_{i,j} = x_{min,j} + rand(0,1) \cdot (x_{max,j} - x_{min,j}) \quad (7)$$

$\mathbf{x}_{min}$ is the source having minimum fitness value and $\mathbf{x}_{max}$ is the source having maximum fitness value at the present iteration. p_i is the probability of accepting solution $\mathbf{x}_i$. The steps of bee colony algorithm is shown below. Let $0 < \tau \leq 1$ be a random constant.

Algorithm: Artificial Bee Colony

```
1.  Initialize a set of solutions $\mathbf{X} = \{\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_n\}$.
2.  Evaluate the fitness of each solution $fit(\mathbf{x}_i)$
    for $1 \leq i \leq n$ and find $\mathbf{x}_{min}$ and $\mathbf{x}_{max}$.
3.  While terminating condition is unsatisfactory do
4.      For each $\mathbf{x}_i \in \mathbf{X}$ do
5.          Compute a new solution by using Eq. (6).
            by randomly selecting $\mathbf{x}_k$.
6.          Evaluate the fitness $fit(\mathbf{x}_i)$.
7.      EndFor
8.      Select a set of some solutions $\mathbf{X}'$ having
        high fitness values.
9.      Compute the probability of selected solutions in
        $\mathbf{X}'$ from Eq. (5).
10.     Select a set of some solutions $\mathbf{X}''$ from $\mathbf{X}'$
        where each selected solution $\mathbf{x}_k \in \mathbf{X}'$ has $p_k > \tau$.
11.     Compute new solution from $\mathbf{X}''$.
12.     Evaluate the fitness of solutions in $\mathbf{X}''$.
13.     Select a set of some solutions $\mathbf{X}'''$ having high
        fitness values.
14.     If the distance between any two solutions is too
        close then
15.       Randomly generate a new solution using Eq. (7).
16.     EndIf
17.     Find $\mathbf{x}_{min}$ and $\mathbf{x}_{max}$ solutions.
18. EndWhile
```

C. Particle Swarm Optimization

This method [7] imitates the behaviour of a particle in a swarm during food search. At time t, each particle i has a solution (position) $\mathbf{x}_i(t) = [x_{i,1}(t), x_{i,2}(t), \ldots, x_{i,d}(t)]^T$, where d is the dimensionality of solution. A set of particles $\mathbf{S}(t) = \{\mathbf{x}_1(t), x_2(t), \ldots, \mathbf{x}_m(t)\}$ is called *swarm*. Each particle i has its own velocity defined as a d-tuple $\mathbf{v}_i(t) = [v_{i,1}(t), v_{i,2}(t), \ldots, v_{i,d}(t)]^T$ of velocity in each dimension. For evaluating the best solution, there are two types of best solutions to focused. The first best solution is the local best solution found with respect to each particle itself. The second best solution is the global best solution found with respect to all particles in the swarm $\mathbf{S}$. Each particle has its own best solution found so far at time t. This best solution is called *personal best* solution. Let $\mathbf{p}_i(t) = [p_{i,1}(t), p_{i,2}(t), \ldots, p_{i,d}(t)]^T$ be the personal best solution of particle i. When considering all particles, the leading particle among all particles has the global best solution. Let $\mathbf{p}_g(t) = [p_{g,1}(t), p_{g,2}(t), \ldots, p_{g,d}(t)]^T$ be the global best solution found at particle g. The velocity and position in dimension j of particle i are updated by the following equations.

$$\begin{aligned} v_{i,j}(t+1) &= w \cdot v_{i,j}(t) + c_1 \cdot r_1 \cdot (p_{i,j} - x_{i,j}(t)) \quad (8) \\ &\quad + c_2 \cdot r_2 \cdot (p_{g,j} - x_{i,j}(t)) \\ x_{i,j}(t+1) &= x_{i,j}(t) + v_{i,j}(t) \quad (9) \end{aligned}$$

Constants $c_1, c_2 \in R$ are weighting factors for personal best and global best solutions, respectively. $r_1, r_2 \in [0,1]$ are random constants to adjust the moving direction of a particle during the search process. w is an inertia weight to control the impact of the past velocity over the current velocity. The algorithm for particle swarm optimization is shown below. The final solution is the *global best* solution.

Algorithm: Particle Swarm Optimization

```
1.  Set $t = 1$.
2.  Initialize a set of solutions $\mathbf{S}(t)$.
3.  Evaluate each solution in $\mathbf{S}(t)$.
4.  Find personal best $\mathbf{p}_i(t)$ and global best $\mathbf{p}_g(t)$.
5.  While $t \leq N$ do
6.      For each solution $\mathbf{x}_i(t) \in \mathbf{S}(t)$ do
7.          Compute new velocity of $\mathbf{x}_i(t)$ using Eq. (8).
8.          Compute new solution $\mathbf{x}_i(t+1)$ from $\mathbf{x}_i(t)$
            using Eq. (9).
9.          Evaluate $\mathbf{x}_i(t+1)$.
10.         Find personal best solution $\mathbf{p}_i(t)$.
11.     EndFor
12.     Find global best solution $\mathbf{p}_g(t)$.
13.     If $t < \delta \times N$ then
14.        Update $\omega = \omega - \frac{\omega_{start} - \omega_{end}}{\delta \times N}$.
15.     EndIf
16.     $t = t + 1$.
17. EndWhile
```

$0 < \delta \leq 1$ is a random constant and N is the maximum number of iterations. The inertia weight ω is in the range of $[\omega_{start}, \omega_{end}]$.

D. Ant Colony Optimization

The objective is to find the shortest path in a graph. It is based on the observation of how an ant finds the best path to reach the food from its current location [9]. An ant deposits some pheromone to tell other ants to follow it. From the graph, a vertex is referred by a location representing a solution of the optimization problem. Each edge (i, j) with is associated with a pheromone variable $\tau_{i,j}$. Let $\mathcal{N}_i$ is the neighbouring vertices of vertex i where ant a is present. At time t, the probability of ant a to move to vertex $j \in \mathcal{N}_i$ is defined as follows.

$$p_{i,j}^{(a)}(t) = \begin{cases} \frac{\tau_{i,j}(t) \cdot d_{i,j}^{-\beta}}{\sum_{k \in \mathcal{N}_i} \tau_{i,k}(t) d_{i,k}^{-\beta}} & \text{if } j \in \mathcal{N}_i \\ 0 & \text{otherwise} \end{cases} \tag{10}$$

$\beta > 0$ is a constant defining the relative importance of pheromone and the distance $d_{i,j}$ between vertices i and j. The amount of pheromone $\tau_{i,j}(t)$ at time $t+1$ is updated by the following equation.

$$\tau_{i,j}(t+1) = (1 - \alpha) \cdot \tau_{i,j}(t) + \sum_{k=1}^{m} \Delta\tau_{i,j}^{(k)} \tag{11}$$

where m is the number of ants and $\Delta\tau_{i,j}^{(k)}$ is defined as follows.

$$\Delta\tau_{i,j}^{(k)} = \begin{cases} L_a^{-1} & \text{if } (i, j) \text{ is in the tour of ant } a \\ 0 & \text{otherwise} \end{cases} \tag{12}$$

L_a is the total tour distance of ant a.

Algorithm: Ant Colony Optimization

```
1. Initialize the values of parameters
2. Repeat
3.     Randomly select a staring vertex for each ant.
4.     For each ant $a$ do
5.         Compute the moving probability and update
           pheromone.
6.         Move ant $a$ to the vertex with the highest
           probability.
7.     EndFor
8. Until the ending condition is satisfied.
```

E. Election Campaign Optimization

The optimization [20] involves a set of candidates, a set of local and global voters, the location of each voter, the location of each candidate, the influence of each candidate on his voters, and the prestige of each candidate. The locations of all candidates are used to compute the best cost function. A candidate c_i is placed at location x_{c_i} and a global voter g_j is placed at location x_{g_j}. Each candidate c_i has a set of local supporters $l_{i,j}$ located at $x_{l_{i,j}}$. The prestige of a candidate c_i is computed by the cost function $f(x_{c_i})$ of the problem whose variable is the location of candidate c_i as follows.

$$p_{c_i} = f(x_{c_i}) \tag{13}$$

Obviously, the value of prestige can be either increased or decreased according to the location of the candidate. The range of area size covering the local and global voters of each candidate c_i can be defined by using the values of prestige and the predefined range of the candidate in the following equations.

$$R_{c_i} = \frac{p_{c_i} - p_{c_i}^{(min)}}{p_{max} - p_{min}}(R_{max} - R_{min}) + R_{min} \tag{14}$$

$$p_{max} = \max_{\forall c_i}(p_{c_i}) \tag{15}$$

$$p_{min} = \min_{\forall c_i}(p_{c_i}) \tag{16}$$

R_{max} and R_{min} are predefined values of maximum range and minimum range, respectively. During the campaign, each candidate will survey the opinions of local sampled voters about his prestige. The deviation local survey of prestige is defined as follows.

$$\delta_{c_i} = \delta_{max} - \frac{p_{c_i} - p_{max}}{p_{max} - p_{min}}(\delta_{max} - \delta_{min}) \tag{17}$$

δ_{max} and δ_{min} are the predefined maximum and minimum deviations, respectively. The location of each global voter x_{g_i} in the range in between x_{max} and x_{min} with the uniform probability u is defined by the following equation.

$$x_{g_i} = x_{max} + u(x_{max} - x_{min}) \tag{18}$$

But the location of each local voter $x_{l_{i,j}}$ of candidate c_i is defined by using normal distribution with the mean at x_{c_i} and standard deviation δ_{c_i} as follows.

$$x_{l_{i,j}} = N(x_{c_i}, \delta_{c_i}^2) \tag{19}$$

Each candidate c_i can influence his voter's decision. The degree of influence on a global voter g_j and a local voter $l_{i,j}$ are defined by the following equations.

$$I_{c_i,g_j} = \begin{cases} \frac{R_{c_i}-d(x_{c_i},x_{g_j})}{R_{c_i}} p_{c_i} & \text{if } R_{c_i} \geq d(x_{c_i},x_{g_j}) \\ 0 & \text{otherwise} \end{cases} \quad (20)$$

$$I_{c_i,l_{i,j}} = \begin{cases} \frac{R_{c_i}-d(x_{c_i},x_{l_j})}{R_{c_i}} p_{c_i} & \text{if } R_{c_i} \geq d(x_{c_i},x_{l_{i,j}}) \\ 0 & \text{otherwise} \end{cases} \quad (21)$$

$d(x_{c_i},x_{g_j})$ and $d(x_{c_i},x_{l_{i,j}})$ are the Euclidean distances between candidate c_i and global voter g_j and local voter $l_{i,j}$, respectively. Each global and local voters have different degree preferences to vote each candidate c_i, which can be measured by these equations.

$$P_{g_j,c_i} = \frac{I_{c_i,g_j}}{\sum_{\forall c_i} I_{c_i,g_j}} f(x_{g_j}) \quad (22)$$

$$P_{l_{i,j},c_i} = \frac{I_{c_i,l_{i,j}}}{\sum_{\forall c_i} I_{c_i,l_{i,j}}} f(x_{l_{i,j}}) \quad (23)$$

These preferences are used further to compute the degree of support from global and local voters for any candidate in the following equation.

$$S_{c_i} = \sum_{\forall g_j} P_{g_j,c_i} + \sum_{\forall g_j} P_{l_{i,j},c_i} \quad (24)$$

Once the candidate knows his degree of support from global and local voters, his moves to the new location toward his voters by using the following equation.

$$x_{c_i} = \sum_{\forall g_j} x_{g_j} \frac{P_{g_j,c_i}}{S_{c_i}} + \sum_{\forall g_j} x_{l_{i,j}} \frac{P_{l_{i,j},c_i}}{S_{c_i}} \quad (25)$$

The new value of cost function is re-computed from this new location x_{c_i}. The computing process to find a set of new locations of candidates with better values of cost function is iterated until a satisfactory value is found.

F. Bat Algorithm

This approach [4], [8] is based on the observation of how a bat exploits and avoids the obstacles to find food. Each bat emits a signal and analyzes the signal echo to avoid obstacles. This behavior involves the signal frequency, loudness of signal, location of bat, and flying velocity. At time t, bat i emits a pulse signal of $f_i^{(t)}$ frequency with loudness $l_i^{(t)}$. The pulse signal has a pulse rate of $p_i^{(t)}$. Bat i is at location $\mathbf{x}_i^{(t)}$ with flying velocity of $\mathbf{v}_i^{(t)}$. For a given problem, the candidate solution corresponding to bat i is evaluated by a cost function $c(\mathbf{x}_i^{(t)})$. Let $f_i^{(t)}$ be the frequency emitted by bat i at time t. The flying velocity and location of bat i at time $t+1$ are computed by the following equations.

$$f_i^{(t)} = f_{min} + (f_{max} - f_{min}) \times rand(0,1) \quad (26)$$

$$\mathbf{v}_i^{(t+1)} = \mathbf{v}_i^{(t)} + (\mathbf{x}_i^{(t)} - \mathbf{x}_{best}) f_i^{(t)} \quad (27)$$

$$\mathbf{x}_i^{(t+1)} = \mathbf{x}_i^{(t)} + \mathbf{v}_i^{(t)} \quad (28)$$

f_{min} and f_{max} are respectively the minimum and maximum frequencies. If the objective is to find the minimum cost value, then $\mathbf{x}_{best}$ at current time is computed by equation (29). Otherwise, $\mathbf{x}_{best}$ is computed by equation (30).

$$\mathbf{x}_{best} = \arg\min_i (c(\mathbf{x}_i^{(t)})) \quad (29)$$

$$\mathbf{x}_{best} = \arg\max_i (c(\mathbf{x}_i^{(t)})) \quad (30)$$

The loudness and pulse rate are temporally adjusted by these equations.

$$l_i^{(t+1)} = \alpha l_i^{(t)} \quad (31)$$

$$p_i^{(t)} = p_i^{(0)}(1 - e^{-\gamma\epsilon}) \quad (32)$$

α and γ are constants and ϵ is a scaling factor. The values of $l_i^{(t)}$ and $p_i^{(t)}$ are in between 0 and 1 and used to select and accept the best solution among the generated solutions. In this bat algorithm, the location $\mathbf{x}_i^{(t)}$ is used to denote the i^{th} solution of the optimization problem. The process of bat algorithm is briefly given as follows.

Algorithm: Bat Algorithm

```
1.  Define a cost function $c(\mathbf{x}_i^{(t)})$ in terms of bat location $\mathbf{x}_i^{(t)}$
    of bat $i$ at time $t$.
2.  Initialize a set of locations of bats with their own
    velocities $\{\mathbf{x}_i^{(0)}, \mathbf{v}_i^{(0)} \mid 1 \leq i \leq n\}$.
3.  Initialize the value of $f_{min}$ and $f_{max}$.
4.  Initialize pulse rate $r_i^{(0)}$ and loudness $l_i^{(0)}$ of each bat.
5.  Define the maximum number of iterations $N$ and set
    $t = 1$.
6.  **While** $t \leq N$ **do**
7.        Generate and update a set of new locations and
          velocities by using equations (26), (27), and (28).
8.        **If** $\exists i\ (rand(0,1) > p_i^{(t)})$ **then**
9.            Find $\mathbf{x}_{best}$.
10.           Generate a new location $\mathbf{x}^{(t)}$ around $\mathbf{x}_{best}$ by
                  $\mathbf{x}^{(t)} = \mathbf{x}_{best} + \epsilon l_i^{(t)}(2 \cdot rand(0,1) - 1)$.
11.       **EndIf**
12.       Randomly generate a new bat at location $\mathbf{x}_j^{(t)}$ and
          velocity $\mathbf{v}_j^{(t)}$, loudness $l_j^{(t)}$, and pulse rate $p_j^{(t)}$.
13.       **If** $(rand(0,1) < l_j^{(t)})$ and $(c(\mathbf{x}_j^{(t)}) < c(\mathbf{x}^{(t)}))$
          **then**
14.           Accept location $\mathbf{x}_j^{(t)}$.
15.           Compute the pulse rate $p_j^{(t+1)}$ and loudness
              $l_j^{(t+1)}$.
16.       **EndIf**
17.       Find the current $\mathbf{x}_{best}$.
18.       $t = t + 1$.
19. **EndWhile**
```

G. Fish Swarm Intelligent Algorithm

The algorithm [1], [21] is based on the following moving behaviours of fish: *random move, search for food, chase for food*, and *follow the swarm*. This imitation of fish behaviour may be stuck at a local best solution. Thus, one extra process called *leap* is attached at the end of algorithm. For a given problem, the solution of the problem is encoded in forms of the location of fish i in a d-dimensional space. Let $\mathbf{x}_i = [x_{i,1}, x_{i,2}, \dots, x_{i,d}]^T$. A cost function $f(\mathbf{x}_i)$ is defined to evaluate the value of solution $\mathbf{x}_i$. A fish selects a moving type by determining the visual result. Each fish has its own visual range or scope. In this algorithm, a visual scope refers to a region around a fish where it looks for food. This region may be empty (no fish), not so crowed (having some fish), or very crowded (full of fish). The determination of moving types of fish i at current location $\mathbf{x}_i$ and its new location is described in the following algorithm.

Algorithm: Fish Swarm Intelligent Algorithm

```
1.  Initialize a swarm of $m$ fish and their locations $\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_m$.
2.  Set $t = 1$.
3.  **While** $t \leq$ maximum number of iterations **do**
4.      **For** each fish and its location $\mathbf{x}_i$ **do**
5.          **If** the visual scope is empty **then**
6.              use *random move* to define a new location $\mathbf{y}_i$.
7.          **EndIf**
8.          **If** the visual scope is crowded **then**
9.              use *search move* to define a new location $\mathbf{y}_i$.
10.         **EndIf**
11.         **If** the visual scope is not so crowded **and**
            the center of swarm gives better result **then**
12.             use *follow-the-swarm move* to define the first tentative location $\mathbf{y}_i^{(1)}$.
13.         **else**
14.             use *search move* to define the first tentative location $\mathbf{y}_i^{(1)}$.
15.         **EndIf**
16.         **If** location $\mathbf{y}_i^{(1)}$ does not give better result
                than the current location $\mathbf{x}_i$
            **then**
17.             use *chase move* to define another new tentative location $\mathbf{y}_i^{(2)}$.
18.         **else**
19.             use *search move* to define another new location $\mathbf{y}_i^{(2)}$.
20.         **EndIf**
21.         Set $\mathbf{y}_i = \arg\min_{\{\mathbf{y}_i^{(1)}), \mathbf{y}_i^{(2)}\}} (f(\mathbf{y}_i^{(1)}), f(\mathbf{y}_i^{(2)}))$.
22.         Set $\forall i \ \mathbf{x}_i = \arg\min_{\{\mathbf{x}_i, \mathbf{y}_i\}} (f(\mathbf{x}_i), f(\mathbf{y}_i))$.
22.         **If** the current result is not better than the
                previous $M$-iterations results **then**
23.             use *leap* process to jump to the new location.
24.         **EndIf**
25.     **EndFor**
26.     $t = t + 1$.
27. **EndWhile**
```

There are lower bound l_j and upper bounds u_j for each dimension j to initialize the value $x_{i,j}$ of each fish i as follows.

$$x_{i,j} = l_j + \alpha_j \cdot rand(0,1) \tag{33}$$

The location $\mathbf{x}_i$ is used to evaluate the result by using a cost function $f(\mathbf{x}_i)$ with $\mathbf{x}_i$ as its variable. The best result is defined by the following equations depending on the objective to achieve minimum (eq. (34)) or maximum (eq. (35)) solution.

$$f_{best} = \min_i (f(\mathbf{x}_i)) \tag{34}$$

$$f_{worst} = \max_i (f(\mathbf{x}_i)) \tag{35}$$

The width w of visual scope for any fish is set by using the lower and upper bounds and a constant parameter ν as follows.

$$w = \nu \cdot \max_{1 \leq j \leq d} (u_j - l_j) \tag{36}$$

This width is used for setting a new location in *random move*. A new location of each type of move is computed in the following equation.

Random move: Let $y_{i,k} \in \mathbf{y}_i$ be a new location $\mathbf{y}_i$ of fish i in dimension k. $\lambda_1, \lambda_2 \in [0,1]$ are two random constants.

$$\mathbf{y}_{i,k} = \begin{cases} x_{i,k} + \lambda_2 \cdot w & \text{if } u_k - x_{i,k} > w \text{ and } \lambda_1 > 0.5 \\ x_{i,k} + \lambda_2 \cdot (u_k - x_{i,k}) & \text{if } u_k - x_{i,k} \leq w \text{ and } \lambda_1 > 0.5 \\ x_{i,k} + \lambda_2 \cdot w & \text{if } x_{i,k} - l_k > w \text{ and } \lambda_1 \leq 0.5 \\ x_{i,k} + \lambda_2 \cdot (x_{i,k} - l_k) & \text{if } x_{i,k} - l_k \leq w \text{ and } \lambda_1 \leq 0.5 \end{cases} \tag{37}$$

Search move: For fish i, a new location $\mathbf{y}_i$ is randomly generated within the visual scope. Fish i moves to this location $\mathbf{y}_i$ if $f(\mathbf{y}_i) < f(\mathbf{y}_i)$. Otherwise, it stays at location $\mathbf{x}_i$.

Follow-the-swarm move: Fish i moves towards the center of visual scope of fish i. Let $\mathbf{V}_i$ be a set of fish in visual scope of fish i. The new location is set to $\mathbf{c}_i$ which is defined in the following.

$$\mathbf{c}_i = \frac{\sum_{\mathbf{x}_k \in \mathbf{V}_i} \mathbf{x}_k}{|\mathbf{V}_i|} \tag{38}$$

Chase move: Fish i moves towards the location $\mathbf{x}_{min}$, whose cost function is minimum, along the moving direction $\mathbf{m}_i = [m_{i,1}, m_{i,2}, \dots, m_{i,d}]^T = \mathbf{x}_{min} - \mathbf{x}_i$. Each new location $y_{i,k}$ is computed as follows.

$$y_{i,k} = \begin{cases} x_{i,k} + \beta \frac{d_{i,k}}{|\mathbf{m}_i|}(u_k - x_{i,k}) & \text{if } m_{i,k} > 0. \\ x_{i,k} + \beta \frac{d_{i,k}}{|\mathbf{m}_i|}(x_{i,k} - l_k) & \text{if } m_{i,k} \leq 0. \end{cases} \tag{39}$$

$\beta \in [0,1]$ is a random constant.

Leap process: Set a new location $y_{new,k}$ in dimension k to avoid being stuck at local cost by the following equations. Let $\lambda_1, \lambda_2 \in [0,1]$ be two random constants.

$$y_{new,k} = \begin{cases} x_{r,k} + \lambda_2 \cdot (u_k - x_{r,k}) & \text{if } \lambda_1 > 0.5 \\ x_{r,k} - \lambda_2 \cdot (x_{r,k} - l_k) & \text{if } \lambda_1 \leq 0.5 \end{cases} \tag{40}$$

$x_{r,k}$ is the value of dimension k of a new randomly selected location $\mathbf{x}_r$ from the initial swarm of fish.

H. Cuckoo Search

The search concept [2], [6], [10] is based on the concept of *Levy* flight. A cuckoo deposits one egg at a time in a selected nest of a host bird. The host bird may get rid of some eggs, including the the eggs of cuckoo, in its nest in order to increase the hatching probability of its eggs. A solution of a given problem is viewed as a host nest. A cuckoo randomly flies to select any nest for laying its eggs and evaluates the quality of nest by a cost function defined in terms of nest variable. Let $\mathbf{x}_i^{(t)} = [x_{i,1}, x_{i,2}, \ldots, x_{i,d}]^T$ be the i solution at iteration t and $f(\mathbf{x}_i^{(t)})$ be the cost function. There are n host nests. The algorithm for cuckoo search is shown as follows.

Algorithm: Cuckoo Search Algorithm

```
1.  Generate a set of initial solutions $\{\mathbf{x}_1^{(0)}, \mathbf{x}_2^{(0)}, \ldots, \mathbf{x}_n^{(0)}\}$.
2.  Set $t = 1$.
3.  **While** *the satisfactory solution is not found* **do**
4.          Randomly pick a solution $\mathbf{x}_i^{(t-1)}$.
5.          Compute each new solution $\mathbf{x}_i^{(t)}$ from $\mathbf{x}_i^{(t-1)}$ by
            applying *Levy* flight.
6.          Evaluate the quality of solution by $f(\mathbf{x}_i^{(t)})$.
7.          Randomly pick a solution $\mathbf{x}_j^{(t-1)}$.
8.          **If** $f(\mathbf{x}_i^{(t)}) > f(\mathbf{x}_j^{(t)})$ **then**
9.              Replace $\mathbf{x}_j^{(t-1)}$ by $\mathbf{x}_i^{(t)}$.
10.         **EndIf**
11.         Randomly delete some fraction of low quality
            solutions and generate new solutions.
12.         Evaluate the quality of all solutions.
13.         Keep some best solutions.
14.         $t = t + 1$.
14. **EndWhile**
15. Pick the best quality solution.
```

Lévy flight was proposed by Paul Pierre Lévy to measure the traveling distance of object in a random walk. In a simple random walk, the traveling distance d_t of an object at time t is defined as a sum of t displacements Δd_t, shown in equation (41).

$$d_t = \sum_{i=1}^{t} \Delta d_i \tag{41}$$

Δd_i is an independent and identically distributed random variable. Lévy flight is similarly described as the simple random walk but each step size Δd_i of displacement is defined by the following probability

$$p_{\Delta d_i}(\beta) = \frac{1}{\Delta d_i^{\beta}} \tag{42}$$

where $1 \leq \beta \leq 3$. In cuckoo search, each step size Δd_i is set equally but it is scaled by the probability $p_{\Delta d_i}(\beta)$. Therefore, the new solution $\mathbf{x}_i^{(t)}$ in step 5 is computed by the concept of Lévy flight as follows.

$$\mathbf{x}_i^{(t)} = \mathbf{x}_i^{(t-1)} + \Delta d \cdot p_{\Delta d_i}(\beta) \tag{43}$$

I. Group Search Optimization

This approach [18] is based on the observation of foraging strategy of some animals living in a group. Each animal has three types of foraging strategies. The first strategy is searching food by itself within a certain direction and a fixed distance. The direction is defined in terms of searching angles with respect to the current location. This strategy is called *producing*. The animal uses the first search strategy is called *a producer*. The second strategy is joining the group to search food. This strategy is called *scrounging* and the animal uses the strategy is called *scrounger*. The last strategy is similar to the first strategy but the direction and distance of food search are random. This strategy is called *dispersing*. The location of each animal in group search is encoded as the solution of optimization problem. Let $\mathbf{x}_i \in R^n$ be the location of animal i at current time. The location of a producer at time t is denoted as $\mathbf{x}_p$. Each location $\mathbf{x}_i$ is evaluated by a cost function cost $f(\mathbf{x}_i)$. Cost $f(\mathbf{x}_i)$ is better than $f(\mathbf{x}_j)$ if $f(\mathbf{x}_i) < f(\mathbf{x}_j)$. The overview of group search optimization is shown as follows.

Algorithm: Group Search Algorithm

```
1.  Initialize the set of locations, $\mathbf{X} = \{\mathbf{x}_i \mid 1 \leq i \leq n\}$.
2.  Compute the cost of each location by $f(\mathbf{x}_i)$.
3.  **While** the stopping conditions are not satisfactory **do**
4.      **For** each $\mathbf{x}_i \in \mathbf{X}$ **do**
5.          Random pick a location $\mathbf{x}_p$ from $\mathbf{X} - \{\mathbf{x}_i\}$
            as the location of producer.
6.          Tentatively compute 3 new locations in 3
            directions: go straight ($\mathbf{x}_{p1}$) , go right ($\mathbf{x}_{p2}$),
            go left ($\mathbf{x}_{p3}$) by using producer strategy
            from $\mathbf{x}_p$.
7.          $\mathbf{x}_{best} = \arg \min_{\mathbf{x}_{p1}, \mathbf{x}_{p2}, \mathbf{x}_{p3}} (f(\mathbf{x}_{p1}), f(\mathbf{x}_{p2}), f(\mathbf{x}_{p3}))$.
8.          **If** $\mathbf{x}_{best} = \mathbf{x}_p$ **then**
9.              Compute new location $\mathbf{x}_{new}$ from $\mathbf{x}_p$ by
                using a new random degree angle and put it
                in $\mathbf{X}$.
10.             Let $\mathbf{y}$ be the producer from past $t$ iterations.
11.             **If** $f(\mathbf{x}_{new}) > f(\mathbf{y})$ **then**
12.                 Replace $\mathbf{x}_{new}$ in $\mathbf{X}$ with $\mathbf{y}$ from past $t$
                    iterations.
13.         **EndIf**
14.         **EndIf**
15.         Randomly select 80% of locations from $\mathbf{X}$ to
            create a set of scroungers $\mathbf{S}$.
16.         Compute new locations for all elements in $\mathbf{S}$
            by using scrounger strategy.
17.         **For** each $\mathbf{x}_j \in \mathbf{X} - \mathbf{S} - \{\mathbf{x}_p\}$.
18.             Compute new location of $\mathbf{x}_j$ by using
                dispersion strategy.
19.         **EndIf**
20.         Compute the cost function of all locations
            in $\mathbf{X}$.
21.     **EndFor**
22. **EndWhile**
```

The new location computed by using producer strategy, scrounger strategy, and dispersion strategy are summarized in the followings.

Producer strategy: A new location in producer strategy is controlled by a directional vector having a angle vector as its variable. Let $\mathbf{d}_i(\theta) = [d_{i,1}, d_{i,2}, \ldots, d_{i,n}]^T$ be a directional vector of location $\mathbf{x}_i$ in n-dimensional space. $\theta = [\theta_1, \theta_2, \ldots, \theta_n]^T$ be an angle vector. Each $d_{i,j}$ is computed from a set of angles $\{\theta_1, \theta_2, \ldots, \theta_n\}$ as follows.

$$d_{i,1} = \prod_{k=1}^{n-1} \cos(\theta_k) \tag{44}$$

$$d_{i,j} = \sin(\theta_{j-1}) \cdot \prod_{k=1}^{n-1} \cos(\theta_k); \quad 2 \le j \le n-1 \tag{45}$$

$$d_{i,n} = \sin(\theta_{n-1}) \tag{46}$$

There are three directions to go for each $\mathbf{x}_i$, i.e. go straight, go right, and go left. Let L be the maximum food pursuit distance and A be the maximum food pursuit angle. The new location of each direction is computed in the followings.

Go straight:

$$\mathbf{x}_{p1} = \mathbf{x}_p + \alpha \cdot L \cdot \mathbf{d}_p(\theta) \tag{47}$$

Go right:

$$\mathbf{x}_{p2} = \mathbf{x}_p + \alpha \cdot L \cdot \mathbf{d}_p(\theta + \frac{A}{2}\mathbf{c}) \tag{48}$$

Go left:

$$\mathbf{x}_{p3} = \mathbf{x}_p + \alpha \cdot L \cdot \mathbf{d}_p(\theta - \frac{A}{2}\mathbf{c}_1) \tag{49}$$

$\alpha \in [0, 1]$ is a random constant and $\mathbf{c}$ is a random constant vector whose each element is in $(0, 1)$.

Scrounger strategy: The new location $\mathbf{x}'_i$ of $\mathbf{x}_i$ in this strategy can be simply computed with respect to the producer $\mathbf{x}_p$ as follows.

$$\mathbf{x}'_i = \mathbf{x}_i + \mathbf{c}_2 \circ (\mathbf{x}_p - \mathbf{x}_i) \tag{50}$$

where $\mathbf{c}_2$ is a random constant vector whose each element is in $(0, 1)$ and the operator $\circ$ refers to Hadamard product.

Dispersion strategy: The new location $\mathbf{x}'_i$ of $\mathbf{x}_i$ in this strategy is governed by a new set of random angles θ' and the current location $\mathbf{x}_i$. This is different from other strategies. The equation is as follows.

$$\mathbf{x}'_i = \mathbf{x}_i + \alpha \cdot \beta \cdot L \cdot \mathbf{d}_i(\theta') \tag{51}$$

$\beta \in (0, 1]$ is a random constant.

J. Brain Storm Optimization

This method [19] is based on the process of brain storming process to solve one particular problem. A group of persons from different or the same background are gathered to discuss the problem and to suggest a good solution to the problem. In terms of evolutionary algorithm, each person is referred to a solution of the problem. Thus, a person can be represented by a vector whose element are the variables of the solution. Let $\mathbf{s}_i$ be solution i being evaluated by a cost function $f(\mathbf{s}_i)$. The best solution $\mathbf{s}_{best}$ is a solution whose $f(\mathbf{s}_{best})$ is minimum.

Algorithm: Brain Storm Algorithm

```
1.  General a set of random solutions $\mathbf{S} = \{\mathbf{s}_1, \mathbf{s}_2, \ldots, \mathbf{s}_n\}$.
2.  While the satisfactory solutions are not found do
3.          Partition $\mathbf{S}$ into $m$ clusters of equal size, $\mathbf{C}_1, \mathbf{C}_2, \ldots, \mathbf{C}_m$.
4.          For each cluster $\mathbf{C}_i$ do
5.              Set its center $\mathbf{c}_i = \arg\min_{\mathbf{s}_j}(f(\mathbf{s}_j))$.
6.          EndFor
7.          Set $p = rand(0, 1)$.
8.          If $p < \tau_1$ then
9.            Randomly select one center $\mathbf{c}_\alpha$.
10.           Replace $\mathbf{c}_\alpha$ with a new randomly generated
              solution $\mathbf{s}_a$.
11.         EndIf
12.         Set $p = rand(0, 1)$.
13.         If $p < \tau_2$ then
14.           Set $p = rand(0, 1)$ and randomly select a
              cluster $\mathbf{C}_\beta$ with center $\mathbf{c}_\beta$.
15.           If $p < \tau_3$ then
16.             Generate a random vector $\mathbf{v}$.
17.             Generate a new solution $\mathbf{s}_a = \mathbf{c}_\beta + \mathbf{v}$.
18.           else
19.             Randomly select $\mathbf{s}_k \in \mathbf{C}_\beta$ and generate a
                random vector $\mathbf{v}$.
20.             Generate a new solution $\mathbf{s}_b = \mathbf{s}_k + \mathbf{v}$.
21.           EndIf
22.         else
23.           Set $p = rand(0, 1)$.
24.           Randomly select two centres $\mathbf{c}_i$ and $\mathbf{c}_j$.
25.           Generate two random vectors $\mathbf{v}$ and $\mathbf{w}$.
. 26.         If $p < \tau_3$ then
27.             Set $\mathbf{c}_k = \mathbf{c}_i + \mathbf{c}_j + \mathbf{v}$.
28.           else
29.             Randomly select solutions $\mathbf{s}_{i_1} \in \mathbf{C}_i$
                and $\mathbf{s}_{j_1} \in \mathbf{C}_j$.
30.             Generate two new solutions $\mathbf{s}_e = \mathbf{s}_{i_1} + \mathbf{v}$
                and $\mathbf{s}_e = \mathbf{s}_{j_1} + \mathbf{w}$.
31.         EndIf
32.         If number of newly generated solutions is not
              equal to $n$ then
33.           Go to step 12.
34.         EndIf
35. EndWhile
```

$\tau_1, \tau_2, \tau_3 \in (0, 1)$ are constants used as the decision probability.

K. Genetic Algorithm

This method [22], [23] imitates the natural process of genetic reproduction. A solution of a problem is encoded in a form of n-bit chromosome. A set of chromosomes is called *population*. Two major genetic reproductions which are mutation and crossover are employed to produce a new generation of population. To reach the satisfactory result, several chromosome reproductions are iterated. Any good chromosomes will be filtered by a cost function to reproduce the next generation but the bad ones are destroyed. Let

$\mathbf{x}_i = [x_{i,1}, x_{i,2}, \ldots, x_{i,n}]^T$ be a chromosome in a form of n-bit vector. Each chromosome is evaluated by a cost function $f(\mathbf{x}_i)$. The overview of genetic algorithm is shown in the following.

Algorithm: Genetic Algorithm

```
1.  Generate an initial set of chromosome population
    $\mathbf{P}^{(1)} = \{\mathbf{x}_i \mid 1 \leq i \leq n\}$.
2.  Evaluate $f(\mathbf{x}_i)$ of each chromosome $\mathbf{x}_i \in \mathbf{P}^{(1)}$.
3.  Set $t = 1$.
4.  **While** terminating conditions are not satisfactory **do**
5.          Select $m$ best chromosomes according to their
            cost functions from $\mathbf{P}^{(t)}$ and put them in $\mathbf{B}$.
6.          Set $\mathbf{P}^{(t)} = \mathbf{P}^{(t)} - \mathbf{B}$.
7.          Use crossover reproduction with some selected
            chromosomes in $\mathbf{P}^{(t)}$ to generate a new set of
            chromosomes $\mathbf{C}$.
8.          Use mutation reproduction with some selected
            chromosomes in $\mathbf{P}^{(t)}$ to generate a new set of
            chromosomes $\mathbf{M}$.
9.          $t = t + 1$.
10.         Set $\mathbf{P}^{(t)} = \mathbf{P}^{(t-1)} \cup \mathbf{C} \cup \mathbf{M}$.
11.         Set $\mathbf{C} = \mathbf{M} = \emptyset$.
12. **EndWhile**
```

The mutation reproduction requires only one chromosome $\mathbf{x}_i$ to randomly produce another bit pattern of chromosome $\mathbf{x}_i$ by flipping one of $(x_{i,1}, x_{i,2}, \ldots, x_{i,n})$ or a subset of $(x_{i,1}, x_{i,2}, \ldots, x_{i,n})$. For example, suppose $\mathbf{x}_i = [11011]^T$. One possibility to mutate $\mathbf{x}_i$ is to flip the bit 0 to 1 to obtain this new bit pattern $[11111]^T$.

On the other hand, the crossover reproduction requires two chromosomes to randomly exchange some bit patterns to obtain two new chromosomes. There are no formal rules of how to select the locations from both chromosomes to perform crossover reproduction. One simple crossover scheme is to randomly select a fixed position as the crossover position in both chromosomes and then exchange the bit patterns to the left and to the right of the crossover position from both chromosomes. For example, suppose chromosomes $(x_{i,1}x_{i,2} \ldots x_{i,\alpha}x_{i,\alpha+1} \ldots x_{i,n})$ and $(x_{j,1}x_{j,2} \ldots x_{j,\alpha}x_{j,\alpha+1} \ldots x_{j,n})$ are selected for performing crossover reproduction. Let α be the crossover position. After crossover reproduction, two new chromosomes are $(x_{j,1}x_{i,2} \ldots x_{j,\alpha}x_{i,\alpha+1} \ldots x_{i,n})$ and $(x_{i,1}x_{i,2} \ldots x_{i,\alpha}x_{j,\alpha+1} \ldots x_{j,n})$.

L. Simulated Annealing

This method [15], [16] is based on the observation of the process of hardening the surface of metal by heating the molecules of metal and then rapidly cool them down. By repeating this process several times, the molecules will rearrange their positions in a very compact structure to harden the surface. Let $\mathbf{x}_i = [x_{i,1}, x_{i,2}, \ldots, x_{i,n}]^T$ be solution i with n variables. These variables are referred to the molecules and the solution is referred to the result of each annealing trial. The performance of each annealing result $\mathbf{x}_i$ is evaluated by a cost function $f(\mathbf{x}_i)$. To select an acceptable solution, a temperature variable T is introduced and used to computed the probability of accepting the solution. The simulated annealing algorithm is shown as follows.

Algorithm: Simulated Annealing Algorithm

```
1.  Initialize the value of temperature $T$, maximum number
    of iterations $M$, and probability threshold $p$.
2.  Set $t = 1$;
3.  Randomly generate a solution $\mathbf{x}_t$.
4.  **While** $T$ is still high **do**
5.          **While** $t \leq M$ **do**
6.                  Generate a new solution $\mathbf{x}'_t$ by perturbing
                    each $x_{t,j}$.
7.                  Compute $\Delta f = f(\mathbf{x}'_t) - f(\mathbf{x}_t)$.
7.                  **If** $\Delta_f < 0$ **then**
8.                      Replace $\mathbf{x}_t$ with $\mathbf{x}'_t$ as a new solution.
9.                  **else**
10.                     **If** probability $e^{-\frac{\Delta f}{T}} > p$ **then**
11.                         Replace $\mathbf{x}_t$ with $\mathbf{x}'_t$ as a solution.
12.                     **EndIf**
13.                 **EndIf**
14.                 $t = t + 1$.
15.         **EndWhile**
16. $T = \alpha \cdot T$.
17. **EndWhile**
```

To reduce the temperature T, the value of α is constantly set as $0 < \alpha < 1$.

III. Generic Structure of Evolutionary Algorithms

As shown in the previous Sections, all algorithms mainly deployed three main processes of randomly generating the solutions, filtering those heuristically unsatisfactory solutions by using a cost function, and randomly regenerating new some solutions either from either the probably good solutions after filtering process or from newly generated solutions. A solution is usually encoded in a forms of vector and interpreted as an animal or human. The process of filtering unsatisfactory solutions and generating new solutions are interpreted as the behavior of animal. Thus, the generic structure of these meta-heuristic algorithm can be established as follows.

Generic Structure of Evolutionary Algorithm

```
1.  Generate a set of solutions $\mathbf{S} = \{\mathbf{s}_1, \mathbf{s}_2, \ldots \mathbf{s}_n\}$.
2.  **While** terminating conditions are not reached **do**
3.          Evaluate each $f(\mathbf{s}_i)$.
4.          Select some best solutions according to $f(\mathbf{s}_i)$
            by its actual value or by a probability value.
5.          Generate some new solutions either from
            the selected solutions or from newly
            generated solutions.
6.  **EndWhile**
```

The terminating conditions may be the maximum number of iterations or the heuristically acceptable $f(\mathbf{s}_i)$. A new solution can be obtained by several methods such as perturbing the solution by adding a random vector to a selected current solution, using random number generator, using mutation concept with the current solution, using crossover concept with two current solutions. From the generic structure, it is noticeable that the evolutionary algorithm has two major groups of steps to reach an acceptable solution. The first one is step 3 which is the evaluation step of solutions. Another one is from steps 4 and 5 for filtering and generating new solution without scrutinizing how the cost function evaluate any solution and the geometrical structure of the cost function in n-dimensional space. Generally, due to the steps 4 and 5, the number of iterations to find an acceptable solution cannot be estimated in advance. This is the disadvantage of evolutionary algorithm although it might produce the best solution in only one iteration.

IV. Opposite Gradient Search on Manifold of Cost Function

This method [24] takes a new approach by searching along the manifold of a cost function. It starts from the cost function $E(\mathbf{x}_i)$ of solution $\mathbf{x}_i$ and views the cost function as a non-linear manifold in a n-dimensional space. A starting location on the manifold is randomly generated as a coordinates $(\mathbf{x}_i, E(\mathbf{x}_i)$. The location of any local maximum or minimum value on the manifold of cost function must have zero gradient, $\nabla E(\mathbf{x}_i) = 0$, and furthermore it must lie in between the positive gradient, $\nabla E(\mathbf{x}_i) > 0$, and negative gradient, $\nabla E(\mathbf{x}_i) < 0$. Figure I shows this observation.

To find the best solution, a set of random solutions is generated as a set of coordinates on the manifold of cost function. Next, two solutions $\mathbf{x}_1$ and $\mathbf{x}_2$ with opposite gradients are randomly selected. The solution with zero gradient must be somewhere in between $\mathbf{x}_1$ and $\mathbf{x}_2$. The next better solution $\mathbf{x}_3$ can be computed by using both $\mathbf{x}_1$ and $\mathbf{x}_2$ and their gradients as follows. Suppose $|\nabla E(\mathbf{x}_1)| > |\nabla E(\mathbf{x}_2)|$.

$$\mathbf{x}_3 = \mathbf{x}_1 + \omega \times \frac{|\nabla E(\mathbf{x}_1)|}{|\nabla E(\mathbf{x}_1)| + |\nabla E(\mathbf{x}_2)|} \times ||\mathbf{x}_1 - \mathbf{x}_2|| \quad (52)$$

where $0 < \omega \leq 1$ is a constant for adjusting the searching speed. An example of searching process is illustrated in Figure II. The subscript of each solution denotes the searching sequence.

The advantage of this method is its searching speed which is much faster than the other methods. but similar to the other methods, this method may be stuck at local minima or maxima due to the set of initial solutions. If the best solution lies within the initial solutions, then the searching steps will eventual reach the best solution.

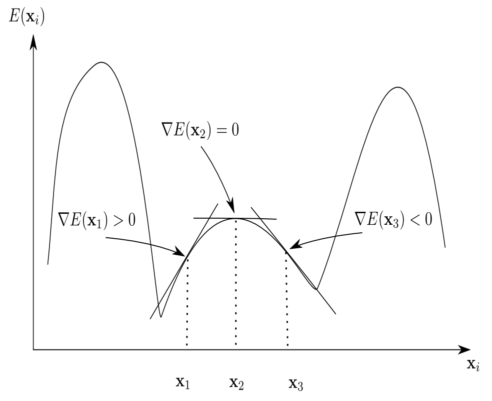
$E(\mathbf{x}_i)$
$\nabla E(\mathbf{x}_2) = 0$
$\nabla E(\mathbf{x}_1) > 0$
$\nabla E(\mathbf{x}_3) < 0$
$\mathbf{x}_1$
$\mathbf{x}_2$
$\mathbf{x}_3$
$\mathbf{x}_i$

Fig. I. The observation of how to find best solution by using the opposite gradients and zero gradient of manifold of the cost function.

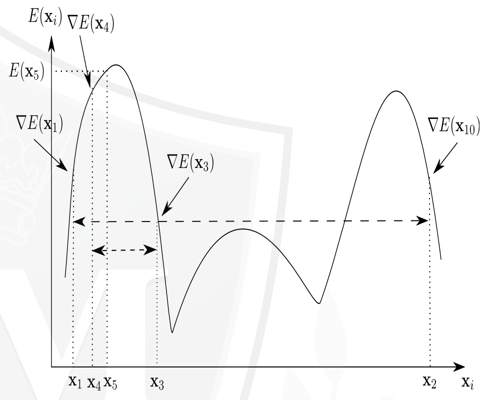

Fig. II. Searching sequence starting from solutions $\mathbf{x}_1$ and $\mathbf{x}_2$.

V. Conclusion

Evolutionary algorithms were proposed to guess the best solution which may or may not be actually the best one but rather be an acceptable one. There were many algorithms in this domain having been introduced so far by adapting the behavior of various types of animal. The disadvantage of these algorithms is due to the lack of considering and examining the geometrical structure of fitness function or cost function as a part of solution generating and finding. The solutions are randomly generated and then evaluated by the fitness function. Some of these solutions are selected by using the value of fitness function with probability and a threshold constant. Although this is a serious disadvantage but it still has an advantage of this approach. A new evolutionary method can be easily developed. Since there is no direct mathematical link between solution generating step and fitness function, it is rather arbitrary to adapt the behavior of new animals with empirical setting of relevant parameters. To overcome the disadvantage, only the approach introduced in [24] transformed a fitness function into a manifold and deployed the observation of where the turning points can be detected by using a fast opposite gradient search. The experiments in [24] signified the faster searching speed and more accurate results than other methods. The research direct should move towards the construction of convex manifold from fitness function and the fast searching step.

References

[1] L. Zhuang, and J. Jiang, “Artificial Fish Swarm Optimization Algorithm Based on Mixed Crossover Strategy”, *International Symposium on Neural Networks ISNN 2013: Advances in Neural Networks? ISNN*, pp. 367-374, 2013.

[2] X. Yang, and S. Deb, "Cuckoo Search via Levy Flights", *2009 World Congress on Nature & Biologically Inspired Computing (NaBIC 2009)*, pp. 210-214, 2009.

[3] I. Zelinka, "A survey on evolutionary algorithms dynamics and its complexity–Mutual relations, past, present and future", *Swarm and Evolutionary Computation*, vol. 25, pp. 2-14, Dec. 2015.

[4] I. Fister Jr., D. Fister, and X. Yang, "A Hybrid Bat Algorithm", *ELEKTROTEHNISKI VESTNIK*, vol. 80(1-2), pp. 1-7, 2013.

[5] D. Karaboga, and B. Basturk, "Artificial Bee Colony (ABC) Optimization Algorithm for Solving Constrained Optimization Problems", P. Melin et al. *(Eds.): IFSA 2007*, LNAI 4529, pp. 789-798, 2007.

[6] I. Fister Jr., D. Fister, and I. Fister, "A comprehensive review of cuckoo search: variants and hybrids", *Int. J. Mathematical Modelling and Numerical Optimisation,* vol. 4, no. 4, pp. 387-409, 2013.

[7] F. Bergh, and A. Engelbrecht, "A Convergence Proof for the Particle Swarm Optimiser", *Fundamenta Informaticae archive,* vol. 105(4), pp. 341-374, Dec. 2010.

[8] X. Yang, "Bat algorithm: literature review and applications", *Int. J. Bio-Inspired Computation*, vol. 5, no. 3, pp. 141-149, 2013.

[9] M. Dorigo, and L. Gambardella, "Ant Colony System: A Cooperative Learning Approach to the Traveling Salesman Problem", *IEEE TRANS- ACTIONS ON EVOLUTIONARY COMPUTATION*, vol. 1, no. 1, Apr. 1997.

[10] P. Civicioglu, and E. Besdok, "A conceptual comparison of the Cuckoo-search, particle swarm optimization, differential evolution and artificial bee colony algorithms", *Artificial Intelligence Review*, vol. 39, no. 4, pp. 315-346, Apr. 2013.

[11] M. Dorigoa, and C. Blumb, "Ant colony optimization theory: A survey", *Theoretical Computer Science,* vol.344, pp. 243-278, Nov. 2005.

[12] S. Das and P. Suganthan, "Differential Evolution: A Survey of the State-of-the-Art", *IEEE TRANSACTIONS ON EVOLUTIONARY COMPUTATION,* vol. 15, no. 1, pp. 4-31. Feb. 2011.

[13] I. Boussad, J. Lepagnot, and P. Siarry, "A survey on optimization metaheuristics", *Information Sciences*, vol. 237, pp. 82-117, Jul. 2013

[14] H. Escalante, M. Montes, and L. Sucar, "Particle Swarm Model Selec- tion", *Journal of Machine Learning Research*, vol. 10, pp. 405-440, 2009.

[15] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi, "Optimization by Simulated Annealing", *Science, New Series,* vol. 220, no. 4598, pp. 671-680, May. 1983.

[16] D. Bertsimass, and J. Tsitsiklis, "Simulated Annealing", *Statistical Science,* vol. 8, no. 1, pp. 10-15, 1993.

[17] S. Mirjalili, and A. Lewis, "The Whale Optimization Algorithm", *Ad- vances in Engineering Software*, vol. 95, pp. 51-67, May. 2016.

[18] S. He, Q. H. Wu, and J. R. Saunders, "Group Search Optimizer: An Optimization Algorithm Inspired by Animal Searching Behavior", *IEEE TRANSACTIONS ON EVOLUTIONARY COMPUTATION*, vol. 13, no. 5, pp. 973-990, Oct. 2009.

[19] Y. Shi, "Brain Storm Optimization Algorithm", Y. Tan et al. *(Eds.): ICSI 2011, Part I, LNCS 6728*, pp. 303-309, 2011.

[20] W. Lva, C. Hea, D. Lia, S. Chenga, S. Luoa, and X. Zhanga, "Election campaign optimization algorithm", *Procedia Computer Science,* vol.1(1) , pp. 1377-1386, May.2012.

[21] E. Fernandes, T. Martins, and A. Rocha, "Fish Swarm Intelligent Algorithm for Bound Constrained Global Optimization", *Proceedings of the International Conference on Computational and Mathematical Methods in Science and Engineering, CMMSE 2009,* 30 June, 1-3 July 2009.

[22] J. Holland, "Genetic Algorithms", *Scientific American*, pp. 66-72, July. 1992.

[23] S. Forrest, "Genetic Algorithms: Principle of Natural Selection Applied to Computation", *Science, New Series*, vol. 261(5123), pp. 872-878, Aug. 1993.

[24] T. Saenphon, S. Phimoltares, and C. Lursinsap, "Combining new Fast Opposite Gradient Search with Ant Colony Optimization for solving traveling salesman problem", *Engineering Applications of Artificial Intelligence,* vol.35, pp. 324-334, Oct. 2014.

Chidchanok Lursinsap received the B. Eng. degree (Hons.) in computer engineering from Chulalongkorn University, Bangkok, Thailand, in 1978, and the M.S. and Ph.D. degrees in computer science from the University of Illinois at Urbana Champaign, Urbana, in 1982 and 1986, respectively. He was a Lecturer with the Department of Computer Engineering, Chulalongkorn University, in 1979. In 1986, he was a Visiting Assistant Professor with the Department of Computer Science, University of Illinois at Urbana-Champaign. From 1987 to 1996, he was with the Center for Advanced Computer Studies, University of Louisiana, Lafayette, as an Assistant and Associate Professor. He then came back to Thailand to establish the Ph.D. program in computer science with Chulalongkorn University and became a Full Professor. His current research interests include neural computing and its applications to other science and engineering areas.