Tutorial on particle swarm optimization and its combinations to other evolutionary algorithms

Loc Nguyen
Loc Nguyen's Academic Network, Vietnam
Email: ng_phloc@yahoo.com
Homepage: www.locnguyen.net

Hassan I. Abdalla
College of Technological Innovation, Zayed University, P.O. Box 144534, Abu Dhabi, United Arab Emirates
Email: Hassan.Abdalla@zu.ac.ae

Ali A. Amer
Computer Science Department, TAIZ University, TAIZ, Yemen
Email: aliaaa2004@yahoo.com

Abstract
Local optimization with convex function is solved perfectly by traditional mathematical methods such as Newton-Raphson and gradient descent but it is not easy to solve the global optimization with arbitrary function although there are some purely mathematical approaches such as approximation, cutting plane, branch and bound, and interval method which can be impractical because of their complexity and high computation cost. Recently, some evolutional algorithms which are inspired from biological activities are proposed to solve the global optimization by acceptable heuristic level. Among them is particle swarm optimization (PSO) algorithm which is proved as an effective and feasible solution for global optimization in real applications. Although the ideology of PSO is not complicated, it derives many variants, which can make new researchers confused. Therefore, this tutorial focuses on describing, systemizing, and classifying PSO by succinct and straightforward way. Moreover, combinations of PSO and other evolutional algorithms for improving PSO itself or solving other advanced problems are mentioned too.
Keywords: particle swarm optimization (PSO), evolutional algorithms, global optimization.

1. Introduction to PSO
Particle swarm optimization (PSO) algorithm was developed by James Kennedy (a social psychologist) and Russell C. Eberhart (an electrical engineer). This tutorial is navigated by the article "Particle swarm optimization: An overview" of Riccardo Poli, James Kennedy, and Tim Blackwell. The main idea of PSO is based on social intelligence when it simulates how a flock of birds search for food. Given a target function known as *cost function* $f(\boldsymbol{x})$, the optimization problem is to find out the minimum point $\boldsymbol{x}^*$ known as minimizer or optimizer so that $f(\boldsymbol{x}^*)$ is minimal. In PSO theory, $f(\boldsymbol{x})$ is also called fitness function and thus, when $f(\boldsymbol{x})$ is evaluated at $f(\boldsymbol{x}_0)$ then, $f(\boldsymbol{x}_0)$ is called fitness value which represents the quality of food source for which a flock of birds search. If $\boldsymbol{x}^*$ is an optimizer, $f(\boldsymbol{x}^*)$ is called optimal value, best value, or best fitness value. As a convention, the optimization problem is global minimization problem when $\boldsymbol{x}^*$ is searched over entire domain of target function $f(\boldsymbol{x})$.

$$\boldsymbol{x}^* = \underset{\boldsymbol{x}}{\operatorname{argmin}} f(\boldsymbol{x})$$

For global maximization, it is simple to change a little bit our viewpoint.

$$\boldsymbol{x}^* = \underset{\boldsymbol{x}}{\operatorname{argmax}} f(\boldsymbol{x})$$

Optimization problem in this research is minimization problem by default if there is no additional explanation. Traditional local optimization methods such as Newton-Raphson and gradient descent along with global optimization methods require that $f(\boldsymbol{x})$ is differentiable. Alternately, PSO does not require existence of differential. In other words, for PSO, $f(\boldsymbol{x})$ can be considered as black box. PSO scatters a population of candidate solutions (candidate optimizers) for $\boldsymbol{x}^*$ and such population is called swarm whereas each candidate optimizer is called particle in the swarm. PSO is an iterative algorithm running over many iterations in which every particle is moved at each iteration so that it approaches the global optimizer $\boldsymbol{x}^*$. Movement of all particles is attracted by $\boldsymbol{x}^*$. In other words, such movement is attracted by minimizing $f(\boldsymbol{x})$ so that $f(\boldsymbol{x})$ is small enough. In PSO, $\boldsymbol{x}$ is considered as position of particle. The movement of each particle is affected by its best position and the best position of the swarm. Note, the closer to $\boldsymbol{x}^*$, the better the position is.

As a formal definition, let $\mathcal{S}$ be the swam of particles and let $\boldsymbol{x}_i$ and $\boldsymbol{p}_i$ be current position and best position of particle i. Note, $\boldsymbol{p}_i$ is called *local best position*. Moreover, the movement speed of particle i is specified by its velocity $\boldsymbol{v}_i$. Let $\boldsymbol{p}_g$ be the *global best position* of entire swarm. The closer to $\boldsymbol{x}^*$, the better the positions $\boldsymbol{p}_i$ and $\boldsymbol{p}_g$ are. It is expected that $\boldsymbol{p}_g$ is equals to $\boldsymbol{x}^*$ or is approximated to $\boldsymbol{x}^*$. The ultimate purpose of PSO is to determine $\boldsymbol{p}_g$.

$$\boldsymbol{p}_g \cong \underset{\boldsymbol{x}}{\operatorname{argmin}} f(\boldsymbol{x})$$

Of course, $\boldsymbol{x}_i$, $\boldsymbol{p}_i$, and $\boldsymbol{p}_g$ are n-dimensional points and $\boldsymbol{v}_i$ is n-dimensional vector because $f(\boldsymbol{x})$ is from real n-dimensional space $\boldsymbol{R}^n$ to real space $\boldsymbol{R}$. Following is pseudo-code of PSO (Wikipedia, 2017).

Input: the swam $\mathcal{S}$ of particles along with their initialized positions and velocities.
Output: the global best position $\boldsymbol{p}_g$ of entire swarm with expectation that $\boldsymbol{p}_g$ is equal or approximated to the global minimizer $\boldsymbol{x}^*$.

Let $\boldsymbol{lb}$ and $\boldsymbol{ub}$ be lower bound and upper bound of particles in their search space. They are vectors. All current positions $\boldsymbol{x}_i = (x_{i1}, x_{i2}, \ldots, x_{in})^T$ of all particles are initialized randomly. Moreover, their best positions are set to be their current positions such that $\boldsymbol{p}_i = \boldsymbol{x}_i$. Note, all particles are randomized in the range $[\boldsymbol{lb}, \boldsymbol{ub}]$ as closed sphere.

$$\boldsymbol{p}_i = \boldsymbol{x}_i \in [\boldsymbol{lb}, \boldsymbol{ub}]$$

All current velocities $\boldsymbol{v}_i$ of all particles are initialized randomly. Because each $\boldsymbol{v}_i$ is vector, its elements are randomized in the range $[-|\boldsymbol{ub} - \boldsymbol{lb}|, |\boldsymbol{ub} - \boldsymbol{lb}|]$ from $-|\boldsymbol{ub} - \boldsymbol{lb}|$ to $|\boldsymbol{ub} - \boldsymbol{lb}|$ where the notation $|.|$ denotes distance between two vectors or two points.

$$\boldsymbol{v}_i = \begin{pmatrix} v_{i1} \\ v_{i2} \\ \vdots \\ v_{in} \end{pmatrix} \text{where } v_{ij} \in [-|\boldsymbol{ub} - \boldsymbol{lb}|, |\boldsymbol{ub} - \boldsymbol{lb}|]$$

The global best position $\boldsymbol{p}_g$ is assigned by the local best position $\boldsymbol{p}_i$ such that $f(\boldsymbol{p}_i)$ is smallest among particles.

$$\boldsymbol{p}_g = \underset{\boldsymbol{p}_i}{\operatorname{argmin}} f(\boldsymbol{p}_i)$$

Repeat (at each current iteration do)
  For every particle i in swarm S
    Velocity of particle i is updated as follows:

$$\boldsymbol{v}_i = \boldsymbol{v}_i + U(0, \phi_1) \otimes (\boldsymbol{p}_i - \boldsymbol{x}_i) + U(0, \phi_2) \otimes (\boldsymbol{p}_g - \boldsymbol{x}_i) \quad (1.1)$$

    Position of particle i is updated as follows:

$$\boldsymbol{x}_i = \boldsymbol{x}_i + \boldsymbol{v}_i \quad (1.2)$$

    If $f(\boldsymbol{x}_i) < f(\boldsymbol{p}_i)$ then
      The best position of particle i is updated: $\boldsymbol{p}_i = \boldsymbol{x}_i$
      If $f(\boldsymbol{p}_i) < f(\boldsymbol{p}_g)$ then

The best position of swarm is updated: $\boldsymbol{p}_g = \boldsymbol{p}_i$
End if
End if
End for
 	
Increase iteration.
Until terminated conditions are met

Table 1.1. Basic particle swarm optimization (PSO) algorithm

From the table above, it is easy to modify a little bit PSO for maximization problem by changing "argmin(.)" to "argmax(.)", changing "If $f(\boldsymbol{x}_i) < f(\boldsymbol{p}_i)$ then" to "If $f(\boldsymbol{x}_i) > f(\boldsymbol{p}_i)$ then", and changing "If $f(\boldsymbol{p}_i) < f(\boldsymbol{p}_g)$ then" to "If $f(\boldsymbol{p}_i) > f(\boldsymbol{p}_g)$ then". If concerning the current t^{th} iteration (the t^{th} time), equation 1.1 and equation 1.2 are denoted as follows:

$$\boldsymbol{v}_i^{(t)} = \boldsymbol{v}_i^{(t-1)} + U(0,\phi_1)\otimes\left(\boldsymbol{p}_i^{(t-1)} - \boldsymbol{x}_i^{(t-1)}\right) + U(0,\phi_2)\otimes\left(\boldsymbol{p}_g^{(t-1)} - \boldsymbol{x}_i^{(t-1)}\right)$$

$$\boldsymbol{x}_i^{(t)} = \boldsymbol{x}_i^{(t-1)} + \boldsymbol{v}_i^{(t)}$$

For general view, PSO is summarized shortly as follows:

Repeat Update velocities and positions of all particles according to equations 1.1 and 1.2. $\boldsymbol{v}_i = \boldsymbol{v}_i + U(0,\phi_1)\otimes(\boldsymbol{p}_i - \boldsymbol{x}_i) + U(0,\phi_2)\otimes\left(\boldsymbol{p}_g - \boldsymbol{x}_i\right)$ $\boldsymbol{x}_i = \boldsymbol{x}_i + \boldsymbol{v}_i$ Reset best positions $\boldsymbol{p}_i$ and $\boldsymbol{p}_g$ for all particles and for entire swarm. Until terminated conditions are met

Table 1.2. Summary of PSO

Equation 1.1 is the heart of PSO, which is called *velocity update rule*. Equation 1.2 is called *position update rule*. There are two most popular terminated conditions:

1. The target function at $\boldsymbol{p}_g$ which is evaluated as $f(\boldsymbol{p}_g)$ is small enough in minimization (or large enough in maximization). For example, $f(\boldsymbol{p}_g)$ is smaller than (or larger than) a threshold.
2. Or PSO ran over a large enough number of iterations.

Function $U(0, \phi_1)$ generates a random vector whose elements are random numbers in the range $[0, \phi_1]$. Similarly, function $U(0, \phi_2)$ generates a random vector whose elements are random numbers in the range $[0, \phi_2]$. For example,

$$U(0,\phi_1) = (r_{11}, r_{12}, \dots, r_{1n})^T \text{ where } 0 \le r_{1j} \le \phi_1$$

$$U(0,\phi_2) = (r_{21}, r_{22}, \dots, r_{2n})^T \text{ where } 0 \le r_{2j} \le \phi_2$$

Note, the super script "^{T}" indicates transposition operator of vector and matrix. The operator $\otimes$ denotes component-wise multiplication of two points (Poli, Kennedy, & Blackwell, 2007, p. 3). For example, given random vector $U(0, \phi_1) = (r_{11}, r_{12},\dots, r_{1n})^T$ and position $\boldsymbol{x}_i = (x_{i1}, x_{i2},\dots, x_{in})^T$, their component-wise multiplication is:

$$U(0,\phi_1)\otimes\boldsymbol{x}_j = \begin{pmatrix} r_{11}x_{i1} \\ r_{12}x_{i2} \\ \vdots \\ r_{1n}x_{in} \end{pmatrix}$$

Two components $U(0,\phi_1)\otimes(\boldsymbol{p}_i - \boldsymbol{x}_i)$ and $U(0,\phi_2)\otimes\left(\boldsymbol{p}_g - \boldsymbol{x}_i\right)$ are considered as attraction forces that push every particle to move. Sources of force $U(0,\phi_1)\otimes(\boldsymbol{p}_i - \boldsymbol{x}_i)$ and force $U(0,\phi_2)\otimes\left(\boldsymbol{p}_g - \boldsymbol{x}_i\right)$ are the particle i itself and its neighbors. Thus, two most important parameters of PSO are ϕ_1 and ϕ_2 which represent the two attraction forces. The popular values of them are $\phi_1 = \phi_2 = 1.4962$. Parameter ϕ_1 along with the force $U(0,\phi_1)\otimes(\boldsymbol{p}_i - \boldsymbol{x}_i)$ express the *exploitation* of PSO whereas parameter ϕ_2 along with the force $U(0,\phi_2)\otimes\left(\boldsymbol{p}_g - \boldsymbol{x}_i\right)$ express the *exploration* of PSO (Poli, Kennedy, & Blackwell, 2007, p. 4). The larger parameter

ϕ_1 is, the faster PSO converges but it trends to converge at local minimizer. In opposite, if parameter ϕ_2 is large, convergence to local minimizer will be avoided in order to achieve better global optimizer but convergence speed is decreased. Parameters ϕ_1 and ϕ_2 are also called acceleration coefficients or *attraction coefficients*. Especially, ϕ_1 is called *cognitive weight* and ϕ_2 is called *social weight* because ϕ_1 reflects thinking of particle itself in moving and ϕ_2 reflects influence of entire swarm on every particle in moving. In practical, velocity $\boldsymbol{v}_i$ can be bounded in the range $[-\boldsymbol{v}_{max}, +\boldsymbol{v}_{max}]$ in order to avoid out of convergence trajectories but the parameter $\boldsymbol{v}_{max}$ is not popular because there are some other parameters such as inertial weight and constriction coefficient (mentioned later) which are used to damp the dynamics of particles. Favorite values for the size of swarm (the number of particles) are ranged from 20 to 50.

Because any movement has inertia, inertial force is added to the two attraction forces. Hence, the inertial force is represented by a so-called *inertial weight* ω where $0 < \omega \leq 1$. Equation 1.1 becomes (Poli, Kennedy, & Blackwell, 2007, p. 4):

$$\boldsymbol{v}_i = \omega \boldsymbol{v}_i + U(0, \phi_1) \otimes (\boldsymbol{p}_i - \boldsymbol{x}_i) + U(0, \phi_2) \otimes (\boldsymbol{p}_g - \boldsymbol{x}_i) \quad (1.3)$$

The larger inertial weight ω is, the faster particles move because its inertial is high, which leads PSO to explore global optimizer. Note that moving fast does not imply fast convergence. In opposite, the smaller ω leads PSO to exploit local optimizer. In general, large ω expresses exploration and small ω expresses exploitation. The inverse $1-\omega$ is known as friction coefficient. The popular value of ω is 0.7298. The equation above is a relatively comprehensive presentation of PSO. The two components $U(0, \phi_1) \otimes (\boldsymbol{p}_i - \boldsymbol{x}_i)$ and $U(0, \phi_2) \otimes (\boldsymbol{p}_g - \boldsymbol{x}_i)$ which are called attraction forces are now called *cognitive component* and *social component* whereas the third component $\omega \boldsymbol{v}_i$ is called *inertial component* (Cleghorn & Engelbrecht, 2014, p. 38).

Pioneers in PSO (Poli, Kennedy, & Blackwell, 2007, p. 5) recognized that if velocities $\boldsymbol{v}_i$ of particles are not restricted, their movements can be out of convergence trajectories at unacceptable levels. Therefore, they proposed a so-called *constriction coefficient* χ to damp dynamics of particles. Note, χ is also called constriction weight or damping weight where $0 < \chi \leq 1$. With support of constriction coefficient, equation 1.1 becomes (Poli, Kennedy, & Blackwell, 2007, p. 5):

$$\boldsymbol{v}_i = \chi \left(\boldsymbol{v}_i + U(0, \phi_1) \otimes (\boldsymbol{p}_i - \boldsymbol{x}_i) + U(0, \phi_2) \otimes (\boldsymbol{p}_g - \boldsymbol{x}_i) \right) \quad (1.4)$$

It is easy to recognize that equation 1.3 is special case of equation 1.4 when the expression $\chi \boldsymbol{v}_i$ is equivalent to the expression $\omega \boldsymbol{v}_i$. The popular value of constriction coefficient is $\chi = 0.7298$ given $\phi_1 = \phi_2 = 2.05$ and $\omega = 1$. Note, inertial weight ω is also the parameter that damps dynamics of particles. This is the reason that $\omega = 1$ when $\chi \neq 1$ but constriction of χ is stronger than ω because χ affects previous velocity $\boldsymbol{v}_i$ and two attraction forces whereas ω affects only previous velocity $\boldsymbol{v}_i$.

Structure of swarm which is determined by defining neighbors and neighborhood of every particle is called *swarm topology* or *population topology*. Because $\boldsymbol{p}_g$ is the best position of entire swarm, the attraction force $U(0, \phi_2) \otimes (\boldsymbol{p}_g - \boldsymbol{x}_i)$ indicates the movement of each particle is affected by all other particles, which means that every particle connects to all remaining particles. In other words, neighbors of a particle are all other particles, which is known as fully connected swarm topology. For easily understandable explanation, suppose particles are vertices of a graph, fully connected swarm topology implies that such graph is fully connected, in which all vertices are connected together. Alternately, swarm topology can be defined in different way so that each particle i only connects with a limit number N_i of other particles. In other words, each particle has N_i neighbors. Suppose there are K groups or K sub-graphs in the swarm, it is possible to keep track of K *grouped best positions* such as $\boldsymbol{p}_{g_1}, \boldsymbol{p}_{g_2}, \ldots, \boldsymbol{p}_{g_K}$ for improving self-seeking ability instead of reserving only one global best position $\boldsymbol{p}_g$. Of course, when PSO finishes, the global best position $\boldsymbol{p}_g$ is selected among such K grouped best positions

$\boldsymbol{p}_{g_1}, \boldsymbol{p}_{g_2},\ldots, \boldsymbol{p}_{g_K}$ or these grouped best positions converge to the global best position $\boldsymbol{p}_g$. As a convention, let $\boldsymbol{p}_g(\boldsymbol{x}_i) = \boldsymbol{p}_{g_k}$ denote the best position of some group k to which a particle i belongs, PSO is modified with self-seeking called grouped PSO, as follows:

Repeat
  For every particle i in swarm S
    Velocity of particle i is updated as follows:

$$\boldsymbol{v}_i = \boldsymbol{v}_i + U(0,\phi_1)\otimes(\boldsymbol{p}_i - \boldsymbol{x}_i) + U(0,\phi_2)\otimes\left(\boldsymbol{p}_g(\boldsymbol{x}_i) - \boldsymbol{x}_i\right) \quad (1.5)$$

    Position of particle i is updated as follows:

$$\boldsymbol{x}_i = \boldsymbol{x}_i + \boldsymbol{v}_i$$

    If $f(\boldsymbol{x}_i) < f(\boldsymbol{p}_i)$ then
      The best position of particle i is updated: $\boldsymbol{p}_i = \boldsymbol{x}_i$
      If $f(\boldsymbol{p}_i) < f(\boldsymbol{p}_g(\boldsymbol{x}_i))$ then
        The best position of the group to which the particle i belongs is updated: $\boldsymbol{p}_g(\boldsymbol{x}_i) = \boldsymbol{p}_i$
      End if
    End if
  End for

  Increase iteration.
Until terminated conditions are met

Table 1.3. Grouped PSO

An ideal terminated condition of grouped PSO is that K grouped best positions $\boldsymbol{p}_{g_1}, \boldsymbol{p}_{g_2},\ldots, \boldsymbol{p}_{g_K}$ converge to the global best position $\boldsymbol{p}_g$. It is easy to deduce that grouped PSO aims to improve the exploration aspect of PSO for global optimizer seeking. Especially, grouped PSO can be more preeminent if it is associated with dynamic topology in which neighbors of particles can change over iterations.

Alternately, Mendes (Poli, Kennedy, & Blackwell, 2007, p. 6) improved self-seeking ability of groups by concerning all neighbors of a particle. Therefore, with custom-defined swarm topology, given particle i connecting with N_i neighbors, Mendes (Poli, Kennedy, & Blackwell, 2007, p. 6) redefined the velocity update rule as follows:

$$\boldsymbol{v}_i = \chi\left(\boldsymbol{v}_i + \frac{1}{N_i}\sum_{k=1}^{N_i} U(0,\phi)\otimes(\boldsymbol{q}_k - \boldsymbol{x}_i)\right) \quad (1.6)$$

Where $\boldsymbol{q}_k$ is the best position of the k^{th} neighbor of particle i. Of course, $\boldsymbol{q}_k$ is $\boldsymbol{p}_j$ of some particle j.

$\boldsymbol{q}_k = \boldsymbol{p}_j$ such that particle j is the k^{th} neighbor of particle i.

Please pay attention that, in equation above, particle i is also its neighbor. In other words, in equation 1.5, the set of N_i neighbors includes particle i. The two parameters ϕ_1 and ϕ_2 are reduced into only one parameter $\phi > 0$, which implies the strengths of all attraction forces from all neighbors on particle i are equal. The popular value of ϕ is $\phi = 4.1$ given $\chi = 0.7298$. This equation is known as Mendes' fully informed particle swarm (FIPS) method. FIPS can be put in grouped PSO with many grouped best positions, or it can keep the global best position $\boldsymbol{p}_g$ as usual.

The topology in the basic PSO specified by equation 1.1, equation 1.3, and equation 1.4 is known *global best topology* because only one best position $\boldsymbol{p}_g$ of entire swarm is kept track. However, equations 1.5 and 1.6 indicate that many best positions from groups implied by grouped best positions or neighbors are kept track. Hence, methods such as grouped PSO and FIPS specify a so-called *local best topology*, which converges slowly but avoids converging at local optimizer. In other words, local best topology aims to exploration rather than exploitation.

If we focus on the fact that the attraction force issued by the particle *i* itself is equivalent to the attraction force from the global best position $\boldsymbol{p}_g$ and the other attraction forces from its neighbors $\boldsymbol{q}_k$, the velocity update rule is modified as follows:

$$\boldsymbol{v}_i = \chi\left(\omega\boldsymbol{v}_i + U(0,\phi_1)\otimes(\boldsymbol{p}_i - \boldsymbol{x}_i) + U(0,\phi_2)\otimes\left(\boldsymbol{p}_g - \boldsymbol{x}_i\right) + \frac{1}{N_i}\sum_{k=1}^{N_i} U(0,\phi)\otimes(\boldsymbol{q}_k - \boldsymbol{x}_i)\right) \quad (1.7)$$

In equation above, the set of K_i neighbors does not include particle *i* and so, the three parameters ϕ_1, ϕ_2, and ϕ are co-existent. Inertial weight ω is kept intact too. It is easy to recognize that equation 1.7 is the general form of velocity update rule. This equation balances local best topology and global best topology with expectation that convergence speed is improved but convergence to local optimizer can be avoided. In other words, this general form aims to achieve both exploration and exploitation. Moreover, this general form can be put in grouped PSO by supporting many grouped best positions as follows:

$$\boldsymbol{v}_i = \chi\left(\omega\boldsymbol{v}_i + U(0,\phi_1)\otimes(\boldsymbol{p}_i - \boldsymbol{x}_i) + U(0,\phi_2)\otimes\left(\boldsymbol{p}_g(\boldsymbol{x}_i) - \boldsymbol{x}_i\right) + \frac{1}{N_i}\sum_{k=1}^{N_i} U(0,\phi)\otimes(\boldsymbol{q}_k - \boldsymbol{x}_i)\right) \quad (1.8)$$

Where $\boldsymbol{p}_g(\boldsymbol{x}_i)$ is the best position of the group to which the particle *i* belongs.

Because the role of constriction weight χ is slightly similar to the role of inertial weight ω, it is possible to set the constriction weight to be 1 so that the general form is simpler.

$$\boldsymbol{v}_i = \omega\boldsymbol{v}_i + U(0,\phi_1)\otimes(\boldsymbol{p}_i - \boldsymbol{x}_i) + U(0,\phi_2)\otimes\left(\boldsymbol{p}_g - \boldsymbol{x}_i\right) + \frac{1}{N_i}\sum_{k=1}^{N_i} U(0,\phi)\otimes(\boldsymbol{q}_k - \boldsymbol{x}_i)$$

In general, the two main aspects of PSO are exploration and exploitation. The exploration aspect aims to avoid premature converging so as to reach global optimizer whereas the exploitation aspect aims to motivate PSO to converge as fast as possible. Besides exploitation property can help PSO to converge more accurately regardless of local optimizer or global optimizer. These two aspects are equally important. Consequently, two problems corresponding to the exploration and exploitation are *premature problem* and *dynamic problem*. Solutions of premature problem are to improve the exploration and solutions of dynamic problem are to improve the exploitation. Inertial weight and constriction coefficient are common solutions for dynamic problem. Currently, solutions of dynamic problem often relate to tuning coefficients which are PSO parameters. Solutions of premature problem relates to increase dynamic ability of particles such as:

- Dynamic topology.
- Change of fitness function.
- Adaptation includes tuning coefficients, adding particles, removing particles, and changing particle properties.
- Diversity control.

Dynamic topology is popular in solving premature problem. The topology mentioned in this section is assumed to be static (Poli, Kennedy, & Blackwell, 2007, p. 6) because it is kept intact over all iterations of PSO. In other words, neighbors and neighborhood in *static topology* are established fixedly. We will later research dynamic topology in which neighbors and neighborhood are changed at each iteration.

2. Variants of PSO

The PSO shown in table 1.1 is basic PSO. Recently there are many PSO variants. Some of them aim to improve the basic PSO but the others aim to solve raised problems.

2.1. Simplified and improved PSOs

Variants of PSO mentioned in this sub-section aim to simplify or improve basic PSO. In general, these variants focus on modifications of basic PSO. Binary PSO (BPSO) developed by James Kennedy is a simple version of PSO where positions $\boldsymbol{x}_i$ and $\boldsymbol{p}_i$ are binary (0 and 1). After velocity update rule (equation 1.1) is executed, the velocity $\boldsymbol{v}_i = (v_{i1}, v_{i2},\ldots, v_{in})^T$ is in turn squashed into range [0, 1] by squash function (logistic function) as follows (Poli, Kennedy, & Blackwell, 2007, p. 9), (Too, Abdullah, & Saad, 2019, p. 3):

$$s(v_{ij}) = \frac{1}{1 + \exp(-v_{ij})}, \forall j \tag{2.1.1}$$

Where v_{ij} is the j^{th} element of $\boldsymbol{v}_i = (v_{i1}, v_{i2},\ldots, v_{in})^T$. Of course, the squashed value $s(v_{ij})$ is in range [0, 1]. The main point of BPSO is to modified position update rule as follows (Too, Abdullah, & Saad, 2019, p. 3):

$$x_{ij} = \begin{cases} 1 \text{ if } s(v_{ij}) > r \\ 0 \text{ if } s(v_{ij}) \leq r \end{cases}, \forall j \tag{2.1.2}$$

Where x_{ij} is the j^{th} element x_{ij} of $\boldsymbol{x}_i = (x_{i1}, x_{i2},\ldots, x_{in})^T$ and r is a random number in range [0, 1]. In general, position update rule in BPSO is specified by equation 2.1.2 instead 1.2. Therefore, the purpose of BPSO is to simplify the basic PSO.

Discrete PSO (DPSO) first proposed by Quan-Ke Pan et al. is another simple version of PSO in which positions $\boldsymbol{x}_i$ are discrete or binary, but it is different from BPSO because it eliminates velocity update rule (equation 1.1). Given two positions $\boldsymbol{x}_i$ and $\boldsymbol{x}_j$ whose elements are nominal or binary, because no arithmetic operator can be executed on them, DPSO alternately uses genetic operators such as exchange operator, one-cut operator, and two-cut operator to determine the position update rule. Without loss of generality, figure 2.1.1 illustrates exchange operator (Guner & Sevkli, 2008, p. 4).

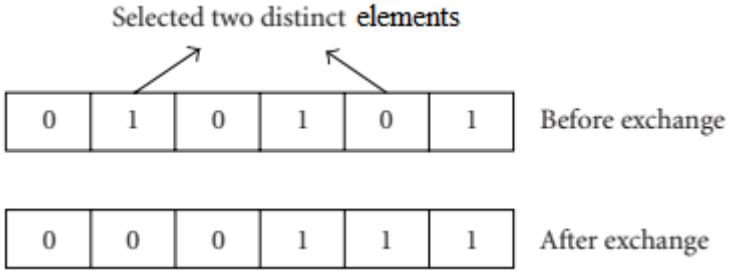

Figure 2.1.1. Illustration of exchange operator

Let $f(\boldsymbol{x}_i, \phi)$ represent exchange operator on $\boldsymbol{x}_i$ with parameter ϕ as probability threshold. Exactly, when elements of $\boldsymbol{x}_i$ are browsed one by one, at each element x_{ij} a new random number r is created with uniform distribution and then, if $r \geq \phi$, exchange operator will be executed on group of such elements x_{ij}. Figure 2.1.2 illustrates one-cut crossover operator (Guner & Sevkli, 2008, p. 5).

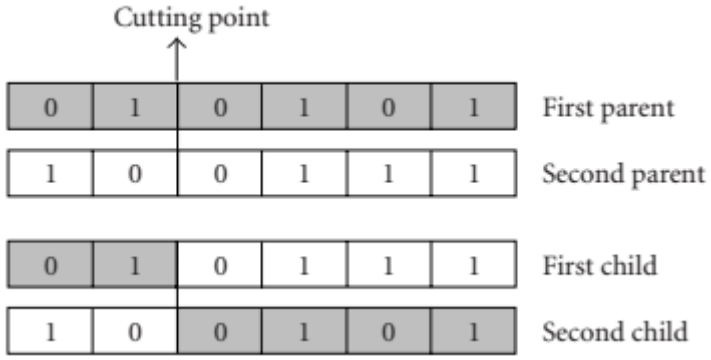

Figure 2.1.2. Illustration of one-cut crossover operator

Let $f_1(\boldsymbol{x}_i, \boldsymbol{p}_i, \phi_1)$ represent one-cut crossover operator on $\boldsymbol{x}_i$ and $\boldsymbol{p}_i$ with parameter ϕ_1 as probability threshold. Exactly, when pairs of $\boldsymbol{x}_i$ and $\boldsymbol{p}_i$ are browsed pair by pair, at each pair (x_{ij}, p_{ij}) a new random number r is created with uniform distribution and then, if $r \geq \phi$, one-cut crossover operator will be executed on group of such pairs (x_{ij}, p_{ij}). Figure 2.1.3 illustrates two-cut crossover operator. It is easy to recognize that $f_1(\boldsymbol{x}_i, \boldsymbol{p}_i, \phi_1)$ plays the role of local attraction force (Guner & Sevkli, 2008, p. 5).

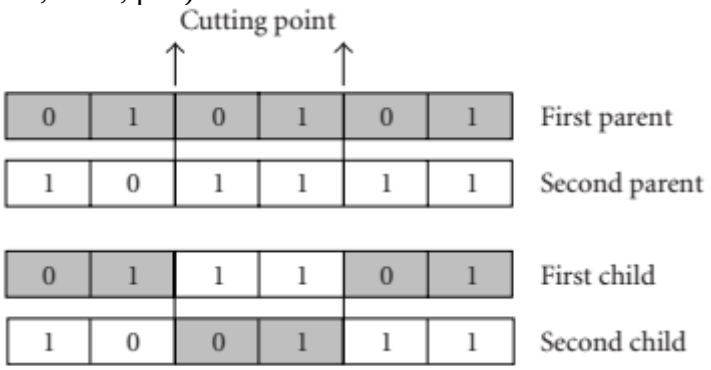

Figure 2.1.3. Illustration of two-cut crossover operator

Let $f_2(\boldsymbol{x}_i, \boldsymbol{p}_g, \phi_2)$ represent two-cut crossover operator on $\boldsymbol{x}_i$ and $\boldsymbol{p}_g$ with parameter ϕ_2 as probability threshold. Exactly, when pairs of $\boldsymbol{x}_i$ and $\boldsymbol{p}_g$ are browsed pair by pair, at each pair (x_{ij}, p_{gj}) a new random number r is created with uniform distribution and then, if $r \geq \phi$, two-cut crossover operator will be executed on group of such pairs (x_{ij}, p_{gj}). It is easy to recognize that $f_2(\boldsymbol{x}_i, \boldsymbol{p}_g, \phi_2)$ plays the role of global attraction force. Finally, the position update rule in DPSO is defined by combinative function of $f(\boldsymbol{x}_i, \phi)$, $f_1(\boldsymbol{x}_i, \boldsymbol{p}_i, \phi_1)$, and $f_2(\boldsymbol{x}_i, \boldsymbol{p}_g, \phi_2)$, as follows (Guner & Sevkli, 2008, p. 4):

$$\boldsymbol{x}_i = f_2\big(f_1(f(\boldsymbol{x}_i, \phi), \boldsymbol{p}_i, \phi_1), \boldsymbol{p}_g, \phi_2\big) \tag{2.1.3}$$

Equation 2.1.3 can be interpreted as an ordered sequence of three equations as follows:

$$\boldsymbol{x}_i = f(\boldsymbol{x}_i, \phi)$$
$$\boldsymbol{x}_i = f_1(\boldsymbol{x}_i, \boldsymbol{p}_i, \phi_1)$$
$$\boldsymbol{x}_i = f_2\big(\boldsymbol{x}_i, \boldsymbol{p}_g, \phi_2\big)$$

Bare bones PSO (BBPSO) developed by James Kennedy and Russell C. Eberhart is also a simple version of PSO where velocity update rule is eliminated. In other words, positions $\boldsymbol{x}_i$ are updated based on only previous position and previous best position. Given $\boldsymbol{x}_i = (x_{i1}, x_{i2}, \ldots, x_{in})^T$, $\boldsymbol{p}_i = (p_{i1}, p_{i2}, \ldots, p_{in})^T$, and $\boldsymbol{p}_g = (p_{g1}, p_{g2}, \ldots, p_{gn})^T$, BBPSO assumes that the j^{th} element x_{ij} of $\boldsymbol{x}_i$ follows normal distribution with mean $(p_{ij}+p_{gj})/2$ and variance $(p_{ij}-p_{gj})^2$ (Poli, Kennedy, & Blackwell, 2007, p. 13).

$$x_{ij} \sim \mathcal{N}\left(\frac{p_{ij} + p_{gj}}{2}, \big(p_{ij} - p_{gj}\big)^2\right), \forall j \tag{2.1.4}$$

Note, the sign "~" denotes distribution and $\mathcal{N}$ denotes normal distribution. Thus, position update rule in BBPSO is modified as follows (Pan, Hu, Eberhart, & Chen, 2008, p. 3), (al-Rifaie & Blackwell, 2012, p. 51):

Every x_{ij} is randomized according to normal distribution

$$\mathcal{N}\left(\frac{p_{ij}+p_{gj}}{2}, \left(p_{ij}-p_{gj}\right)^2\right) \quad (2.1.5)$$

Obviously, position update rule in BBPSO is specified by equation 2.1.4 instead equation 1.2 and there is no velocity update rule. Therefore, the purpose of BPSO is to simplify the basic PSO.

Quantum PSO (QPSO) was proposed by Jun Sun, Bin Feng, and Wenbo Xu, in which movement of particles obeys quantum motion instead of Newton motion. Sun et al. assumed that each particle i is pulled by its local attractor denoted $\boldsymbol{a}_i$ so that $\boldsymbol{a}_i$ moves on the hyperrectangle formed by the local best position $\boldsymbol{p}_i$ and global best position $\boldsymbol{p}_g$ as follows (Fu, Liu, Zhang, & Deng, 2013, p. 2):

$$\boldsymbol{a}_i = \varphi \boldsymbol{p}_i + (1-\varphi)\boldsymbol{p}_g \quad (2.1.6)$$

Where φ is the parameter such that $0 < \varphi < 1$. According to the stochastic movement of the local attractors, velocity update rule is removed from QPSO and position update rule is changed as follows (Fu, Liu, Zhang, & Deng, 2013, p. 2):

$$\boldsymbol{x}_i = \begin{cases} \boldsymbol{a}_i + \beta(M - \boldsymbol{x}_i)\log\frac{1}{u}, \text{if } h > 0.5 \\ \boldsymbol{a}_i - \beta(M - \boldsymbol{x}_i)\log\frac{1}{u}, \text{if } h \leq 0.5 \end{cases} \quad (2.1.7)$$

Where β as a parameter is called contraction-expansion coefficient which is used to control the convergence speed. Besides, h and u are two random numbers which are randomized in runtime process such that $0 < h < 1$ and $0 < u < 1$. Moreover, M is the mean of all local best positions over the population $\mathcal{S}$, as follows (Fu, Liu, Zhang, & Deng, 2013, p. 3):

$$M = \frac{1}{|\mathcal{S}|}\sum_{i=1}^{|\mathcal{S}|}\boldsymbol{p}_i \quad (2.1.8)$$

Note, $|\mathcal{S}|$ is the size of population $\mathcal{S}$ which is the number of particles in $\mathcal{S}$.

Because important parameters known as coefficients such as cognitive weight ϕ_1, social weight ϕ_2, inertial weight ω, and constriction coefficient χ affect on velocity update rule, there are some researches aiming to modifying them at each iteration or each time point so as to improve PSO overtime. If such approach focuses on acceleration coefficients which are cognitive weight ϕ_1, social weight ϕ_2, it is called time-varying acceleration coefficient (TVAC) approach or TVAC-PSO. If a method aims to modify other coefficients like ω and χ, it can be classified into TVAC approach because there is relationship between ω, χ, ϕ_1 and ϕ_2. However, other methods related to adjust other parameters such as population size and topology do not belong to TVAC approach. Here we focus on TVAC-PSOs which can also be solutions for the premature problem and so they can be considered as dynamic PSOs. Shi and Eberhart modified inertial weight ω as follows (Ratnaweera, Halgamuge, & Watson, 2004, p. 241):

$$\omega = (\omega_1 - \omega_2)\frac{t}{T} + \omega_2 \quad (2.1.9)$$

Note t is current iteration and T is the maximum number of iterations in PSO process whereas ω_1 and ω_2 are initial value and final value of the inertial weight belonging to the interval (0, 1). Ratnaweera et al. modified acceleration coefficients φ_1 and φ_2 as follows (Ratnaweera, Halgamuge, & Watson, 2004, p. 242):

$$\varphi_1 = (\varphi_{1f} - \varphi_{1g})\frac{t}{T} + \varphi_{1g}$$
$$\varphi_2 = (\varphi_{2f} - \varphi_{2g})\frac{t}{T} + \varphi_{2g} \quad (2.1.10)$$

Whereas φ_{1f}, φ_{1g}, φ_{2f} and φ_{2g} are constants belonging to the interval (0, 1).

Beside changing acceleration coefficients, changing inertial weight ω in real time is also concerned with some researches. Recall that the larger inertial weight is, the faster particles move, which increases particles dynamics. Therefore, decreasing inertial weight will keep avoiding premature convergence to local optimizer. Let ω_{min} and ω_{max} be minimum inertial weight and maximum inertial weight, respectively, and let t and T be current iteration and the maximum number of iterations, Chatterjee and Siarry (Bonyadi & Michalewicz, 2017, p. 28) modified inertial weight in real time by following formula:

$$\omega = \omega_{min} + \left(\frac{T-t}{T}\right)^n (\omega_{max} - \omega_{min}) \quad (2.1.11)$$

Where n is used to speed up the process of decreasing inertial weight. In general, modifying coefficient parameters is not only cheap but also effective for improving the exploration of PSO.

2.2. Dynamic PSO

As aforementioned, two main problems of PSO are premature problem and dynamic problem. Solutions of premature problem are to improve the exploration so that PSO is not trapped in local optimizer. Exactly, these solutions relate to increase dynamic ability of particles such as dynamic topology, change of fitness function, adaptation (tuning coefficients, adding particles, removing particles, changing particle properties), and diversity control over iterations. As a convention, the solutions for premature problem derive so-called *dynamic PSOs*. This sub-section list popular dynamic PSOs. Note that QPSO and TVAC-PSO aforementioned can be considered as dynamic PSOs although they are not mentioned here again because their modifications for solving premature problem are not emphasized.

Recall that the aforementioned topology is static topology because it is kept intact over all iterations of PSO. Recall that topology implies groups of particles inside swarm population. For easily understandable explanation, suppose particles are vertices of a graph and a connection between two vertices indicates the interaction or the neighborhood of two particles so that such graph is the easily understandable representation of topology. If the graph is fully connected, the topology is called global topology (gbest topology) which is typically specified by equations 1.1, 1.3, and 1.4. If the graph is separated into many fully connected sub-graphs, the topology is called local topology (lbest topology) which is typically specified by equations 1.5, 1.6, and 1.7. Here we research *dynamic topology* in which neighbors and neighborhood are changed at each iteration.

Sugnathan (Poli, Kennedy, & Blackwell, 2007, p. 8) proposed to start PSO with small local best topology with a small number of neighbors and such topology is progressively enlarged with a larger number of neighbors after each iteration until getting the full connected topology known as global best topology (gbest topology). The favorite local best topology is lattice ring.

Peram (Poli, Kennedy, & Blackwell, 2007, p. 8) defined the topology dynamically at each iteration by a so-called fitness distance ratio (FDR). Given target particle i and another particle j, their FDR is the ratio of the difference between $f(\boldsymbol{x}_i)$ and $f(\boldsymbol{x}_j)$ to the Euclidean difference between $\boldsymbol{x}_i$ and $\boldsymbol{x}_j$.

$$\mathrm{FDR}(\boldsymbol{x}_i, \boldsymbol{x}_j) = \frac{|f(\boldsymbol{x}_i) - f(\boldsymbol{x}_j)|}{|\boldsymbol{x}_i - \boldsymbol{x}_j|} \quad (2.2.1)$$

Given target particle i, if FDR($\boldsymbol{x}_i$, $\boldsymbol{x}_j$) is larger than a threshold (> 1), the particle j is a neighbor of the target particle i. Alternately, top K particles whose FDR (s) with $\boldsymbol{x}_i$ are largest are K neighbors of particle i.

Clerc (Poli, Kennedy, & Blackwell, 2007, p. 8) proposed a so-called TRIBES algorithm which is a dynamic topology PSO. TRIBES divides the entire swarm into sub-populations called "tribes". A good tribe has good particles whose best values (best fitness values) $f(\boldsymbol{p}_i)$ are small enough whereas a bad tribe has bad particles whose best values (best fitness values) $f(\boldsymbol{p}_i)$ are not small enough. After some iterations, a good tribe can remove particles that are not good enough in order to maintains its preeminence and a bad tribe can add more particles to increase its possibility of improvement. Because TRIBES adds and removes dynamically particles, it can be classified into adaptation solution for dynamic problem.

Recall that premature problem is solved by many solutions such as dynamic topology, change of fitness function, adaptation (tuning coefficients, adding particles, removing particles, changing particle properties), and diversity control over iterations. Here we research the ways which change fitness function evaluation. Noisy adder is an interesting way to change how to evaluate fitness function, but fitness function is indeed not changed. For instance, any time at any iteration when $f(\boldsymbol{x})$ is evaluated, a random noise is added into the evaluated result.

$$f(\boldsymbol{x}) + \varepsilon$$

This means that the task of resetting best positions $\boldsymbol{p}_i$ and $\boldsymbol{p}_g$ for every particle and for entire swarm in PSO is modified as follows:

```
Randomize two noises ε_{x_i} and ε_{p_i}
If f(x_i) + ε_{x_i} < f(p_i) + ε_{p_i} then
    The best position of particle i is updated: p_i = x_i
    Randomize noise ε_{p_g}
    If f(p_i) + ε_{p_i} < f(p_g) + ε_{p_g} then
        The best position of swarm is updated: p_g = p_i
    End if
End if
```

Table 2.2.1. Resetting best positions with random noise

The noise ε often conforms normal distribution with mean 0 and small enough variance. When $\boldsymbol{x}_i$ is evaluated more than one time, evaluated results can be different. Consequently, it is possible to avoid converging local optimizer while the effectiveness in convergence is kept (Poli, Kennedy, & Blackwell, 2007, p. 10).

As aforementioned, *diversity control* is a solution of premature problem to prevent PSO from converging to local optimizer because a reason of local trapping is that many particles are clustered too tight into one region. Diversity control implies how to disperse particles far away from others in order to decrease the probability that particles are moved near together. Hence, we research here the diversity control. For instance, Lovbjerg and Krink (Poli, Kennedy, & Blackwell, 2007, p. 13) used a so-called self-organized criticality technique to control the diversity of particles. Every particle is associated with a so-called criticality variable and hence, the criticality variable will be increased anytime the associated particle is near to another particle. The particle whose associated criticality variable is larger than a threshold will be diverted from its neighbors by resetting its position. PSO with diversity control by Lovbjerg and Krink (Poli, Kennedy, & Blackwell, 2007, p. 13) can be explained as follows:

```
Repeat
    Update velocities and positions of all particles according to equations 1.1 and 1.2.
        v_i = v_i + U(0, φ_1) ⊗ (p_i − x_i) + U(0, φ_2) ⊗ (p_g − x_i)
        x_i = x_i + v_i
    Reset best positions p_i and p_g for all particles and for entire swarm.
```

For each particle i in swarm
  Let c_i be the associated criticality variable of particle i. Initialize c_i=0.
  For each particle $j \neq i$ in swarm
    If the distance $|\boldsymbol{x}_i - \boldsymbol{x}_j|$ is smaller than a predefined threshold then increasing c_i.
  End for

  If c_i is larger than a predefined threshold then resetting $\boldsymbol{x}_i$ such that $\boldsymbol{x}_i$ becomes a random position.
End for
Until terminated conditions are met

Table 2.2.2. PSO with diversity control

It is possible to deduce that diversity control is related to dynamic topology.

2.3. Multi-objective PSO

Target function $f(\boldsymbol{x})$ in normal optimization problem is scalar-by-vector function whose input $\boldsymbol{x}_0$ and output $f(\boldsymbol{x}_0)$ are vector and scalar, respectively. However, multi-objective optimization requires that target function $\boldsymbol{f}(\boldsymbol{x})$ is vector-by-vector function whose input $\boldsymbol{x}_0$ and output $\boldsymbol{f}(\boldsymbol{x}_0)$ are vector and vector, respectively. In other words, we have:

$$\boldsymbol{f}(\boldsymbol{x}) = \begin{pmatrix} f_1(\boldsymbol{x}) \\ f_2(\boldsymbol{x}) \\ \vdots \\ f_m(\boldsymbol{x}) \end{pmatrix}$$

Such that

$$\boldsymbol{x}^* = \underset{\boldsymbol{x}}{\operatorname{argmin}}\, \boldsymbol{f}(\boldsymbol{x}) = \underset{\boldsymbol{x}}{\operatorname{argmin}} \left(\left(f_1(\boldsymbol{x}), f_2(\boldsymbol{x}), \ldots, f_m(\boldsymbol{x}) \right)^T \right)$$

Or

$$\boldsymbol{x}^* = \underset{\boldsymbol{x}}{\operatorname{argmax}}\, \boldsymbol{f}(\boldsymbol{x}) = \underset{\boldsymbol{x}}{\operatorname{argmax}} \left(\left(f_1(\boldsymbol{x}), f_2(\boldsymbol{x}), \ldots, f_m(\boldsymbol{x}) \right)^T \right)$$

As a convention, there are m partial target functions $f_1(\boldsymbol{x})$, $f_2(\boldsymbol{x})$,…, $f_m(\boldsymbol{x})$ whose input is n-dimension variable $\boldsymbol{x} = (x_1, x_2, \ldots, x_n)^T$. Multi-objective optimization problem needs to be solved for optimizing many targets at the same time and so multi-objective PSO aims to solve multi-objective optimization problem. Multi-objective optimization problem is here multi-objective minimization problem by default. Because $\boldsymbol{f}(\boldsymbol{x})$ is vector function, it is impossible to use the comparison operator for determining minimal value. Therefore, Pareto dominance approach is applied to solve this hazard. Given target vector-by-vector function $\boldsymbol{f}(\boldsymbol{x})$ along with two variables $\boldsymbol{u}$ and $\boldsymbol{v}$, according to Pareto dominance approach for minimization (Yu, Gao, & Wang, 2020), $\boldsymbol{f}(\boldsymbol{u})$ *dominates* $\boldsymbol{f}(\boldsymbol{v})$ if $f_i(\boldsymbol{u}) \leq f_i(\boldsymbol{v})$ for all i and there exists at least an index j such that $f_j(\boldsymbol{u}) < f_j(\boldsymbol{v})$.

$$\boldsymbol{f}(\boldsymbol{u}) < \boldsymbol{f}(\boldsymbol{v}) \overset{\text{def}}{=} \forall i \in \overline{1,m}, f_i(\boldsymbol{u}) \leq f_i(\boldsymbol{v}) \text{ and } \exists j \in \overline{1,m}, f_j(\boldsymbol{u}) < f_j(\boldsymbol{v}) \quad (2.3.1)$$

The notation "<" denotes the concept "domination". Inversely, the event $\boldsymbol{f}(\boldsymbol{u})$ does not dominate $\boldsymbol{f}(\boldsymbol{u})$ is determined as follows:

$$\boldsymbol{f}(\boldsymbol{u}) \not< \boldsymbol{f}(\boldsymbol{v}) \overset{\text{def}}{=} \exists i \in \overline{1,m}, f_i(\boldsymbol{u}) > f_i(\boldsymbol{v}) \text{ or } \forall j \in \overline{1,m}, f_j(\boldsymbol{u}) \geq f_j(\boldsymbol{v}) \quad (2.3.2)$$

Note, the event $\boldsymbol{f}(\boldsymbol{v})$ is *nondominated* by $\boldsymbol{f}(\boldsymbol{u})$ is the same to the event $\boldsymbol{f}(\boldsymbol{u})$ does not dominate $\boldsymbol{f}(\boldsymbol{v})$.

$$\boldsymbol{f}(\boldsymbol{v}) \not> \boldsymbol{f}(\boldsymbol{u}) \Leftrightarrow \boldsymbol{f}(\boldsymbol{u}) \not< \boldsymbol{f}(\boldsymbol{v})$$

If $\boldsymbol{f}(\boldsymbol{u})$ dominates $\boldsymbol{f}(\boldsymbol{v})$ then $\boldsymbol{u}$ is called dominating variable. If $\boldsymbol{f}(\boldsymbol{v})$ is nondominated by $\boldsymbol{f}(\boldsymbol{u})$ (also $\boldsymbol{f}(\boldsymbol{u})$ does not dominate $\boldsymbol{f}(\boldsymbol{v})$) then $\boldsymbol{v}$ is called nondominated variable. The set of nondominated variables is called Pareto optimal set (or optimal set P in short).

$$P = \{\boldsymbol{x} | \boldsymbol{x} \text{ is nondominated}\} = \{\boldsymbol{x} | \forall \boldsymbol{u} \neq \boldsymbol{x}: \boldsymbol{f}(\boldsymbol{u}) \not< \boldsymbol{f}(\boldsymbol{x})\} \quad (2.3.3)$$

Of course, we also have:

$$P = \{\boldsymbol{x} | \nexists \boldsymbol{u} \neq \boldsymbol{x}: \boldsymbol{f}(\boldsymbol{u}) < \boldsymbol{f}(\boldsymbol{x})\}$$

For multi-objective optimization, PSO extracts the optimal set P from the best positions of particles $\boldsymbol{p}_i$. Therefore, the optimal set P in PSO is the set of nondominated positions. Consequently, PSO randomly selects a nondominated position $\boldsymbol{p}_{nd}$ in the optimal set P and then updates velocities based on such nondominated position $\boldsymbol{p}_{nd}$ as follows:

Let B be the set of the best positions of particles as $B = \{\boldsymbol{p}_1, \boldsymbol{p}_2, \boldsymbol{p}_3, \ldots\}$.
Extract the optimal set P from B.

Repeat
  Select randomly a nondominated position $\boldsymbol{p}_{nd}$ in the optimal set P as a pseudo global best position.
  Update velocities and positions of all particles with note that each $\boldsymbol{p}_i$ is obtained from B.

$$\boldsymbol{v}_i = \boldsymbol{v}_i + U(0, \phi_1) \otimes (\boldsymbol{p}_i - \boldsymbol{x}_i) + U(0, \phi_2) \otimes (\boldsymbol{p}_{nd} - \boldsymbol{x}_i) \quad (2.3.4)$$

$$\boldsymbol{x}_i = \boldsymbol{x}_i + \boldsymbol{v}_i$$

  Initialize the new set B' to be the duplicate of the set B, for instance $B' = B$.
  If there exists an index k such that $f_k(\boldsymbol{p}_i) > f_k(\boldsymbol{x}_i)$ which means that $\boldsymbol{x}_i$ is nondominated by $\boldsymbol{p}_i$ then
    Let $\boldsymbol{p}_{old} = \boldsymbol{p}_i$.
    Assign $\boldsymbol{p}_i = \boldsymbol{x}_i$.
    Re-add the old $\boldsymbol{p}_{old}$ to B such that $B = B \cup \{\boldsymbol{p}_{old}\}$.
  End if

  Update the optimal set P from B.
  Reset $B = B$'.
Until terminated conditions are met

Table 2.3.1. Multi-objective PSO

Of course, the best position $\boldsymbol{p}_g$ is selected among the optimal set after PSO finishes. An ideal terminated condition is that nondominated positions in P converge to the best position $\boldsymbol{p}_g$. Let B be the set of best positions $\boldsymbol{p}_i$ of particles, it is easy to extract/update the optimal set P from B by brute-force searching derived from the equation of nondomination as follows:

For each $\boldsymbol{p}_j$ in B (loop 1)
  If $f_i(\boldsymbol{p}_j) = f_i(\boldsymbol{p}_k)$ for all $i \in \overline{1, m}$ and all $k \neq j$ then
    Add $\boldsymbol{p}_j$ to P as $P = P \cup \{\boldsymbol{p}_j\}$.
    Remove $\boldsymbol{p}_j$ from B as $B = B \setminus \{\boldsymbol{p}_j\}$.
  Else
    For each $\boldsymbol{p}_k \neq \boldsymbol{p}_j$ in B (loop 2)
      If there exists an index i such that $f_i(\boldsymbol{p}_k) > f_i(\boldsymbol{p}_j)$ then
        Add $\boldsymbol{p}_j$ to P as $P = P \cup \{\boldsymbol{p}_j\}$.
        Remove $\boldsymbol{p}_j$ from B as $B = B \setminus \{\boldsymbol{p}_j\}$.
        Break out this most inner loop (loop 2).
      End if
    End for
  End if
End for

Table 2.3.2. Brute-force searching for optimal set

Please read the research paper by Yu, Gao, and Wang (Yu, Gao, & Wang, 2020, p. 4) to comprehend the technique to determine the optimal set. For improving multi-objective PSO, instead of randomly selecting a nondominated position $\boldsymbol{p}_{nd}$ for velocity update rule, Qiu et al.

(Qiu, Wang, & Zuo, 2013, p. 826) applied clustering methods to grouping the optimal set into clusters represented by their centers. As a result, they select the nondominated position $\boldsymbol{p}_{nd}$ based on these representative centers (or presentative positions) along with their probabilities for velocity update rule.

2.4. Constrained PSO

If there is requirement of constraints for optimization tasks, the traditional optimization problem becomes constrained optimization problem which is defined as follows:

$$\boldsymbol{x}^* = \underset{\boldsymbol{x}}{\operatorname{argmin}} f(\boldsymbol{x})$$
$$\text{subject to } g_i(x) \leq 0 \text{ where } i = \overline{1,m}$$

Where $g_i(\boldsymbol{x})$ are nonlinear or linear functions called constraint functions. In theory of analytic convex optimization, Lagrange function is composed of the convex target function $f(\boldsymbol{x})$ and constraint functions $g_i(\boldsymbol{x})$ associated with Lagrange multipliers $\lambda = (\lambda_1, \lambda_2, \ldots, \lambda_m)^T$ such as $l(\boldsymbol{x}, \lambda) = f(\boldsymbol{x}) + \sum_{i=1}^{m} \lambda_i g_i(\boldsymbol{x})$. When the Lagrange function $l(\boldsymbol{x}, \lambda)$ is function of $\boldsymbol{x}$ and λ then, minimizing $l(\boldsymbol{x}, \lambda)$ with regard to $\boldsymbol{x}$ is paralleled with maximizing $l(\boldsymbol{x}, \lambda)$ with regard to λ according to theory of duality so that the optimizer $(\boldsymbol{x}^*, \lambda^*)^T$ is saddle point of $l(\boldsymbol{x}, \lambda)$. However, optimizing an arbitrary target function $f(\boldsymbol{x})$ requires the constrained PSO to have a heuristic mechanism. Therefore, Parsopoulos and Vrahatis (Parsopoulos & Vrahatis, 2002, p. 216) replaced the Lagrange function by a so-called penalty function $F(\boldsymbol{x})$ defined as follows:

$$F(\boldsymbol{x}) = f(\boldsymbol{x}) + h(k)H(\boldsymbol{x}) \quad (2.4.1)$$

Note, $h(k)$ is a dynamically modified penalty value where k is the current iteration of PSO whereas $H(\boldsymbol{x})$ is a penalty factor, which will be mentioned later. Consequently, the constrained PSO will optimize the penalty function $F(\boldsymbol{x})$ instead, as usual for PSO.

$$\boldsymbol{x}^* = \underset{\boldsymbol{x}}{\operatorname{argmin}} F(\boldsymbol{x}) \Leftrightarrow \begin{cases} \boldsymbol{x}^* = \underset{\boldsymbol{x}}{\operatorname{argmin}} f(\boldsymbol{x}) \\ \text{subject to } g_i(x) \leq 0 \text{ where } i = \overline{1,m} \end{cases}$$

The ideology is clear but the new problem here is how to define the penalty value $h(k)$ and the penalty factor $H(\boldsymbol{x})$. The penalty value $h(k)$ implies penalty over time, which is proportional to the current iteration, for instances (Parsopoulos & Vrahatis, 2002, p. 219):

$$h(k) = k$$
$$h(k) = k\sqrt{k}$$
$$h(k) = k^2$$

The penalty factor $H(\boldsymbol{x})$ implies penalty over position $\boldsymbol{x}$ given constraints $g_i(\boldsymbol{x})$ (Parsopoulos & Vrahatis, 2002, p. 216).

$$H(\boldsymbol{x}) = \sum_{i=1}^{m} \theta\big(q_i(\boldsymbol{x})\big) q_i(\boldsymbol{x})^{\gamma(q_i(\boldsymbol{x}))} \quad (2.4.2)$$

Where $q_i(\boldsymbol{x})$ set the basic penalty value for each constraint $g_i(\boldsymbol{x})$ (Parsopoulos & Vrahatis, 2002, p. 217).

$$q_i(\boldsymbol{x}) = \max\{q_i(\boldsymbol{x}), 0\}, \forall i = \overline{1,m} \quad (2.4.3)$$

Obviously, if a constraint $g_i(\boldsymbol{x})$ is satisfied as $g_i(\boldsymbol{x}) \leq 0$ then its basic penalty value is zero, $q_i(\boldsymbol{x}) = 0$. The function $\theta(.)$ is multi-stage assignment function which is used to partition a real value into stages, for example, given real value $y = q_i(\boldsymbol{x})$ will be partitioned into three stages $y_1 < y_2 < y_3$ with three corresponding real outputs $a_1 < a_2 < a_3$ as follows (Parsopoulos & Vrahatis, 2002, p. 218):

$$\theta(y) = \begin{cases} a_1 \text{ if } y \leq y_1 \\ a_2 \text{ if } y_1 < y \leq y_2 \\ a_3 \text{ if } y_2 < y \end{cases}$$

The function $\gamma(.)$ is the power of penalty function which implies seriousness of penalty value. It is slightly similar to the multi-stage assignment function, but its output is the power number for power function, for example (Parsopoulos & Vrahatis, 2002, p. 218):

$$\gamma(y) = \begin{cases} i \text{ if } y \leq y_1 \\ j \text{ if } y_1 < y \leq y_2 \\ k \text{ if } y_2 < y \end{cases}$$

Where i, j, and k are integers such that $i < j < k$, for example. Both $\theta(.)$ and $\gamma(.)$ implies seriousness of penalty value but $\theta(.)$ indicates linear seriousness whereas $\gamma(.)$ indicates nonlinear seriousness.

Recall that two main aspects of PSO are exploration and exploitation. Exploitation is as important as exploration because it asserts success of convergence and speed of convergence. Some methods like BPSO and BBPSO improve the exploitation whereas the others like FDR and TRIBES improve the exploration with notice that the exploration for global convergence seemingly receives more concerns. Thus, some algorithms mentioned in next section which are combinations of PSO and other algorithms aim to improve exploitation and exploration.

3. PSO and other algorithms

Although PSO is an effective algorithm for optimization problem, it also has some weak points, for example, the premature problem. Therefore, this section skims combinations of PSO to improve PSO or focus on solving specific problems.

3.1. With other evolutionary algorithms

Evolutionary algorithms (EA) are the ones which simulate natural activities in biological world. PSO is an EA because it simulates how a flock of birds search for food and so its combination to other EA(s) is natural. Some combinations are so tight that it is possible to form hybrid PSOs which are considered as variants of PSO mentioned in previous section. Therefore, this section focuses on flexible combinations of PSO and other EA(s).

Genetic algorithm (GA) (Wikipedia, Genetic algorithm, 2002) is classified into evolutionary approach when it simulates biological activities including natural hybridization and selection. In other words, GA considers computational problems as biological activities and then, solves such problems by imitating biological activities too. GA is designed for not only optimization solving but also other learning algorithms, especially supervised learning. For instance, given a problem whose candidate solutions compose a population, GA codes each solution candidate as a chromosome which consists of genes. Concretely, a chromosome represents a solution candidate, and a gene represents a property of such solution candidate with note that any solution has a set of properties. GA will perform so-call biological operators on chromosomes in order to reach a good enough solution. These biological operators simulating biological activities includes crossover operator, mutation operator, etc. In GA literature, a chromosome is also considered as an individual of population; exactly, each individual owns a chromosome. Essentially, given an initialized population of individuals, GA, which is an iterative algorithm, will produce a generation of new individuals at each iteration by performing biological operators on current individuals of the population. The generation of new individuals will be added to the population at each iteration so that the population get larger and larger. Another important thing in GA is to define the fitness of a given individual (chromosome) because GA performs biological operators only on the individuals whose fitness values are good enough. In order words, GA selects good enough individuals for making biological operators. How to define fitness function depends on concrete applications. GA goes through many iterations until best solution is reached, population is large enough, or fitness level (maybe fitness average) over entire population is good enough. Following is pseudo code of GA (Mallawaarachchi, 2017).

Initialize the population of individuals with note that chromosome of each individual is coded according to properties of solution candidate. Repeat Compute fitness values for individuals (chromosomes) of populations. Select individuals whose fitness values are good enough. Perform biological operators (crossover, mutation, etc.) on the selected individuals to produce new individuals called offspring(s). The set of the offspring(s) called generation are added to the entire population. Util terminated conditions are met.

Table 3.1.1. General GA

Recall that the terminated conditions can be that best solution is reached, the population is large enough, or fitness average over entire population is good enough. Biological operators like crossover, mutation, etc. are executed on selected individuals but exactly, such operators are executed on selected chromosomes because each individual has its own chromosome. Recall that each chromosome is a set of genes where each gene represents a property of solution. In other words, each chromosome represents an individual. For easy explanation, a chromosome here is coded by an array of bit (whose values are 0 and 1) and so each bit codes a gene. In real applications, bits can be indices.

1	0	1	0	1	0	1	0	1

Figure 3.1.1. Simple coding chromosome

Because crossover and mutation are two most important operators, they are described shortly here. Crossover, which is binary operator, exchanges mutually genes of two origin individuals called parents to create a new offspring whose chromosome inherits genes of its both parents. Task of exchanging genes occurs at a so-called crossover point.

1	1	1	0	1	0	1	0	1
0	0	0	1	0	1	0	1	0
				↓				
1	1	1	1	0	1	0	1	0

Figure 3.1.2. Crossover operator

For extending GA, crossover operator is any two-operand operator that combines two genes.

Mutation, which is simpler unary operator, changes randomly some genes of one individual so as to create a new offspring.

1	0	1	0	1	0	1	0	1
				↓				
0	1	0	0	1	0	1	0	1

Figure 3.1.3. Mutation operator

For extending GA, mutation operator is any one-operand operator that changes a gene itself.

Recall that there are two most important points in GA, in which the first one is to code chromosomes and perform biological operators on chromosomes whereas the second one is to define the fitness function. Exactly, the fitness function implies the way to calculate fitness values of chromosomes (individuals) and so maybe it is not algebraic function. In other words, a fitness value implies evaluation of a solution. For easy explanation, fitness function can be inverse of target function $-f(\boldsymbol{x}_i)$ in minimization or target function itself $f(\boldsymbol{x}_i)$ in maximization. There is no explicit parameter in GA except that how to code chromosome can be considered as implicit parameter.

As usual, the combination of GA and PSO is performed by applying GA into PSO, in which PSO particles are considered as GA individuals and hence, particle positions are considered as

chromosomes. Because elements of particle $\boldsymbol{x}_i$ are real numbers, biological operators like crossover and mutation are executed based on indices of these elements. For example, given two position $\boldsymbol{x}_1 = (56, 67, 34, 45)^T$ and $\boldsymbol{x}_2 = (12, 23, 78, 89)^T$ and given crossover point 2, the offspring $\boldsymbol{x}_3$ created from crossover operator can be $\boldsymbol{x}_3 = (56, 67, 78, 89)^T$. Wang et al. (Wang, Wang, Xu, Qin, & Qin, 2019) applied GA into PSO for rescheduling high-speed railway timetables, in which they could define the fitness function as inverse of the target function – $f(\boldsymbol{x}_i)$ in minimization or the target function itself $f(\boldsymbol{x}_i)$ in maximization. Before updating velocities of particles, Wang et al. (Wang, Wang, Xu, Qin, & Qin, 2019, pp. 5-6) produced new particles by made crossover operator and mutation operator on particles because they considered these PSO particles as GA individuals (GA chromosome). Following is their modification for velocity update rule (Wang, Wang, Xu, Qin, & Qin, 2019, p. 6).

Compute fitness values for particles. Select particles whose fitness values are good enough. Perform biological operators on the selected particles to produce new particles. Add new particles to the swarm. Make velocity update rule for every particle by equation 1.1. $\boldsymbol{v}_i = \boldsymbol{v}_i + U(0, \phi_1) \otimes (\boldsymbol{p}_i - \boldsymbol{x}_i) + U(0, \phi_2) \otimes (\boldsymbol{p}_g - \boldsymbol{x}_i)$

Table 3.1.2. Velocity update rule for GA based PSO

Note, position update rule is kept intact.

Ant bee colony (ABC) algorithm proposed by Derviş Karaboğa simulates how a colony of bees search for food sources (Wikipedia, Artificial bee colony algorithm, 2007). There are three types of bees such as employed bees, onlooker bees and scout bees. Each employed bee is associated with its position $\boldsymbol{x}_i = (x_{i1}, x_{i2}, \ldots, x_{in})^T$ which in turn represents a food source as a candidate solution for optimizing the target function. The nectar amount of a food source $\boldsymbol{x}_i$ is called fitness value $f_{\boldsymbol{x}_i}$ which can be evaluated as inverse of the target function $f_{\boldsymbol{x}_i} = -f(\boldsymbol{x}_i)$ in minimization or the target function itself $f_{\boldsymbol{x}_i} = f(\boldsymbol{x}_i)$ in maximization. After finding food sources, all employed bees share their nectar amounts with onlooker bees so that onlookers select the best food source – the best position $\boldsymbol{x}_g$ whose fitness value $f_{\boldsymbol{x}_g}$ is best. The selection strategy can be a roulette wheel selection mechanism which is based on the largest solution probability of nectar amounts. An employed bee becomes a scout if it cannot find out a new source after some unsuccessful trials of finding food sources; at that time such scout will reset its position. In general, ABC which is an iterative algorithm has many iterations in which each iteration has three phase such as employed phase, onlooker phase, and scout phase. The terminated condition can be that the overall fitness value is good value, or the number of iterations is large enough. Following is the pseudo-code of ABC (Karaboga & Akay, 2009, pp. 113-114):

The N-size population of employed bees and onlooker bees is initialized. Each employed bee has a random position $\boldsymbol{x}_i$. Repeat *Employed phase*: each employed bee looks a new position $\boldsymbol{x}_i$' regarding other employed bees. $x'_{ik} = x_{ik} + r_1(x_{ik} - x_{jk})$ (3.1.1) If the new fitness value (nectar amount) $f_{\boldsymbol{x}'_i}$ is larger than the old one $f_{\boldsymbol{x}_i}$, such employed bee moves to $\boldsymbol{v}_i$. $\boldsymbol{x}_i = \boldsymbol{x}'_i$ if $f_{\boldsymbol{x}'_i} > f_{\boldsymbol{x}_i}$ (3.1.2)

Onlooker phase: onlooker bees select the i^{th} food source which is the i^{th} solution based on nectar amounts which are modeled as selection probability P_i as follows:

$$P_i = \frac{f_{\boldsymbol{x}_i}}{\sum_{j=1}^{N} f_{\boldsymbol{x}_j}} \quad (3.1.3)$$

Therefore, which g^{th} food source has the largest P_g is selected. Of course, the fitness value of the g^{th} food source is $f_{\boldsymbol{x}_g}$.

Scout phase: If an employed bee cannot find out better food source, for example, $f(\boldsymbol{x}_i) > f_{\boldsymbol{x}_g}$ after some iterations then, it becomes a scout. At that time, such scout reset its position as follows:

$$\boldsymbol{x}_i = \boldsymbol{lb} + r_2(\boldsymbol{ub} - \boldsymbol{lb}) \quad (3.1.4)$$

Until some terminated condition is met.

Table 3.1.3. Ant bee colony (ABC) algorithm

In equation 3.1.1, indices k and j are randomized whereas r_1 is a random number in interval [–1, 1]. In equation 3.1.4, $\boldsymbol{lb}$ and $\boldsymbol{ub}$ are lower bound and upper bound of search space, respectively whereas r_2 is a random number in interval [0, 1].

There are some researches which combined PSO and ABC. Sharma et al. (Sharma, Pant, & Abraham, 2013) proposed a so-called Local Global variant Artificial Bee Colony (LGABC) that puts PSO information into ABC, which aims to balance the exploration and exploitation in ABC when ABC achieves higher exploitation than PSO because ABC updates new positions in employed phase more randomly than PSO whereas PSO achieves higher exploration than ABC because PSO updates new positions based on best local positions and best global positions. Concretely, Sharma et al. proposed a controlled parameter called modification rate (*MR*) and modified the position update equation 3.1.1 in employed phase as follows (Sharma, Pant, & Abraham, 2013, pp. 2-3):

$$\begin{cases} x'_{ik} = x_{ik} + r_1\left(x_{ik} - x_{jk}\right) \text{ if } r \le MR \\ \boldsymbol{x}'_i = \boldsymbol{x}_i + \boldsymbol{v}_i \text{ otherwise} \end{cases} \quad (3.1.5)$$

Where $\boldsymbol{v}_i$ is PSO velocity update rule specified by equation 1.1:

$$\boldsymbol{v}_i = \boldsymbol{v}_i + U(0, \phi_1) \otimes (\boldsymbol{p}_i - \boldsymbol{x}_i) + U(0, \phi_2) \otimes \left(\boldsymbol{p}_g - \boldsymbol{x}_i\right)$$

Note, r is random number in interval [0, 1] and the parameter *MR* is predefined in open interval (0, 1). Simply, the ABC position update equation 3.1.5 becomes PSO position update equation 1.2 if $r > MR$.

3.2. With machine learning algorithms

Machine learning algorithms mentioned in this section are not the evolutionary algorithms related directly to biological activities, but they also simulate or imitate phenomena in real world. The first machine learning algorithm which is mentioned is different evolution algorithm because it can be classified into group of evolutionary algorithms.

Different evolution (DE) algorithm (Wikipedia, Differential evolution, 2005), which is like PSO, is used to solve the optimization problem but it is inclined to find local optimizer like traditional methods such as Newton-Raphson and gradient descent.

$$\boldsymbol{x}^* = \underset{\boldsymbol{x}}{\operatorname{argmin}} f(\boldsymbol{x})$$

Like PSO, DE does not require existence of differential and so, $f(\boldsymbol{x})$ can be considered as black box. Like gradient descent method, DE which is an iterative algorithm improves "pseudo gradient" after every iteration. The term "pseudo gradient" which is not a true gradient is essentially a simulation of analytic differential. Exactly, DE tries to calculate the discrete simulation of differential of $f(\boldsymbol{x})$ with random numbers and then moves toward such pseudo gradient to reach good enough optimizer and hence, such pseudo gradient is improved

progressively. This is the reason that it is called different evolution algorithm. In general, its ideology is slightly similar to gradient descent method. For easy explanation, given two points (a, c) and (b, x), the pseudo gradient or pseudo differential d can be explained as follows:

$$d = \frac{x - c}{b - a}$$

This implies:

$$x = c + d(b - a)$$

DE keeps the pseudo differential d intact as a parameter. In DE literature, d is called differential weight. DE pushes x according to the differential weight d along with random numbers a, b, and c until x gets optimal. In general, particles $\boldsymbol{x}_i$ in PSO are called agents in DE with suppose that each agent $\boldsymbol{x}_i = (x_{i1}, x_{i2},\ldots, x_{in})^T$ is n-dimension point. The swarm $\mathcal{S}$ in PSO is called population $\mathcal{S}$ in DE. There are two main parameters in DE such as the differential d and a so-called crossover probability r which is a number in interval (0, 1). The crossover probability r indicates the chance that DE moves $\boldsymbol{x}_i$ towards the differential weight d with combinations of random numbers a, b, and c. For extension, given agent $\boldsymbol{x}_i = (x_{i1}, x_{i2},\ldots, x_{in})^T$, given three vectors $\boldsymbol{a} = (a_1, a_2,\ldots, a_n)^T$, $\boldsymbol{b} = (b_1, b_2,\ldots, b_n)^T$, and $\boldsymbol{c} = (c_1, c_2,\ldots, c_n)^T$, and given differential weight d and crossover probability r, a new derived agent $\boldsymbol{x} = (x_1, x_2,\ldots, x_n)^T$ is produced by differential crossover as follows:

$$x_j = \begin{cases} c_j + d(b_j - a_j) \text{ if } r_j < r \text{ or } j = k \\ x_{ij} \text{ otherwise} \end{cases}, \forall j \qquad (3.2.1)$$

Where k is a random index in $\{1, 2,\ldots, n\}$ and r_j is a random number in interval (0, 1). Given differential weight d and crossover probability r, DE which is based on differential crossover is described as follows (Wikipedia, Differential evolution, 2005):

Repeat
  For each agent $\boldsymbol{x}_i = (x_{i1}, x_{i2},\ldots, x_{in})^T$ in $\mathcal{S}$
    Pick randomly three vectors $\boldsymbol{a} = (a_1, a_2,\ldots, a_n)^T$, $\boldsymbol{b} = (b_1, b_2,\ldots, b_n)^T$, and $\boldsymbol{c} = (c_1, c_2,\ldots, c_n)^T$ in $\mathcal{S}$ so that they are mutually distinguished and different from $\boldsymbol{x}_i$ too.
    Pick a random index k in $\{1, 2,\ldots, n\}$ where n is dimension of $\boldsymbol{x}_i$.

    Let $\boldsymbol{x} = (x_1, x_2,\ldots, x_n)^T$ be a new agent arbitrarily initialized.
    For j=1 to n
      Pick a random number r_j in interval (0, 1).
      If $r_j < r$ or $j=k$ then set $x_j = c_j + d(b_j - a_j)$ otherwise set $x_j = x_{ij}$.
    End for
    If $\boldsymbol{x}$ is better than $\boldsymbol{x}_i$, for instance ($f(\boldsymbol{x}) < f(\boldsymbol{x}_i)$ in minimization or $f(\boldsymbol{x}) > f(\boldsymbol{x}_i)$ in maximization, then replacing $\boldsymbol{x}_i$ by $\boldsymbol{x}$ as $\boldsymbol{x}_i = \boldsymbol{x}$.
  End for
Until terminated conditions are met

Table 3.2.1. General DE

The terminated conditions can be that there are a good enough agent $\boldsymbol{x}_i$ occurring in the population $\mathcal{S}$ or the number of iterations is large enough. It is easy to recognize that the semantic meaning of differential weight in DE is similar to the semantic meaning of velocity in PSO although DE does not mention biological activities. Therefore, it is possible to classify DE into group of evolutionary algorithms although DE is here categorized into group of machine learning algorithms. Anyhow, evolutionary programming is sub-domain of machine learning.

About combination of SA and PSO, Yu et al. (Yu, Cao, Shan, Zhu, & Guo, 2014) switched mutually DE differential crossover and PSO velocity-position update rules for updating particle positions. The mutual switching is based on a random number. Following (Yu, Cao, Shan, Zhu, & Guo, 2014, p. 5) is their new velocity-position update rules with differential weight d and crossover probability r.

Pick a random number p.
If p is smaller than a predefined threshold then
  Pick randomly three vectors $\boldsymbol{a} = (a_1, a_2,\ldots, a_n)^T$, $\boldsymbol{b} = (b_1, b_2,\ldots, b_n)^T$, and $\boldsymbol{c} = (c_1, c_2,\ldots, c_n)^T$ in $\mathcal{S}$ so that they are mutually distinguished and different from the current position $\boldsymbol{x}_i = (x_{i1}, x_{i2},\ldots, x_{in})^T$ too.
  Pick a random index k in $\{1, 2,\ldots, n\}$ where n is dimension of $\boldsymbol{x}_i$.

  Let $\boldsymbol{x} = (x_1, x_2,\ldots, x_n)^T$ be a new position arbitrarily initialized.
  For j=1 to n
    Pick a random number r_j in interval (0, 1).
    If $r_j < r$ or $j=k$ then set $x_j = c_j + d(b_j - a_j)$ otherwise set $x_j = x_{ij}$.
  End for
Else
  Update velocity $\boldsymbol{v}_i$ and position $\boldsymbol{x}_i$ as usual according to equations 1.1 and 1.2.

$$\boldsymbol{v}_i = \boldsymbol{v}_i + U(0, \phi_1)\otimes(\boldsymbol{p}_i - \boldsymbol{x}_i) + U(0, \phi_2)\otimes(\boldsymbol{p}_g - \boldsymbol{x}_i)$$

$$\boldsymbol{x}_i = \boldsymbol{x}_i + \boldsymbol{v}_i$$

End if

Table 3.2.2. Velocity-position update rules by DE-PSO switching

The goal of the DE-PSO switching is to improve exploration ability of PSO by giving more choices for moving particles. In other words, dynamics of particles is increased.

Simulated annealing (SA) (Wikipedia, Simulated annealing, 2003), which is one of popular heuristic algorithms for solving hazard optimization problems, simulates the process of metal annealing in which a metal bar is firstly heated at the first and highest temperature and later on, such metal bar is cooled gradually by decreasing progressively temperature. SA is designed for not only optimization solving but also other learning algorithms. SA is an iterative algorithm where it finds better solutions at each iteration so as to reach the best solution or a good enough solution; essentially SA is a cooling process which stops at concrete temperature at which a good enough solution is reached. Candidate solutions in SA are called states. Therefore, at each iteration SA decreases the annealing temperature and then, randomizes a new state and calculates energy of such new state. Consequently, SA will decide whether the new state is a good enough candidate solution or not based on a so-called acceptance probability which is a function whose inputs are the current temperature and the energies of the new state and the current state. Note, energy of a state implies thermodynamic free energy of such state. Energy here is similar to entropy which measures thermodynamic turbulence of particles at a state when a metal bar is healed at concrete temperature. The cooler the temperature is, the smaller the energy is. Let s and s_{new} be the current state and new state, respectively and let $E(s)$ and $E(s_{new})$ be their energies, respectively. Let T be the current temperature and let $P(E(s), E(s_{new}), T)$ be the acceptance probability whose inputs are s, s_{new}, and T. If $P(E(s), E(s_{new}), T)$ is larger than a threshold which is often a random number then, s_{new} is set to be the current state as $s = s_{new}$. Following is pseudo code of SA.

Initialize current temperature T by highest temperature T_0 as $T = T_0$.

Repeat
  Decrease current temperature, for example, T = decrease(T).
  Select a random neighbor of current state as s_{new} = neighbor(s).
  If $P(E(s), E(s_{new}), T)$ is larger than a predefined threshold (or a random number) then
    $s = s_{new}$
  End if
Util terminated conditions are met.

Table 3.2.3. General SA

The terminated conditions can be that best state (solution) is reached, the current state *s* is good enough, the current temperature *T* is low enough, or the number of iterations is large enough. Recall that states in SA are candidate solutions. The explicit parameter of SA is the first temperature T_0. It is necessary to describe shortly three important functions such as temperature decreasing function decrease(*T*), neighbor selection function neighbor(*s*), and acceptance probability $P(E(s), E(s_{new}), T)$. A usual, given a maximum iteration number *K* and the current iteration number *k*, the temperature decreasing function can be defined as follows:

$$\text{decrease}(T) = T - \frac{k}{K}T \tag{3.2.2}$$

The acceptance probability is a mechanism for SA to decide if accepting a new state as current state (current candidate solution). The larger the acceptance probability is, the more likely the new state is accepted. Moreover, the acceptance probability is inversely proportional to current temperature *T*. The cooler the current temperature is, the larger the acceptance probability is. According to the Kirkpatrick method, the acceptance probability is defined by exponential function as follows:

$$P(E(s), E(s_{new}), T) = \begin{cases} 1 \text{ if } E(s_{new}) < E(s) \\ \exp\left(-\dfrac{E(s_{new}) - E(s)}{T}\right) \text{ otherwise} \end{cases} \tag{3.2.3}$$

There is an important question about how to define the energy $E(s)$. Because state s represents a candidate solution in SA, its energy $E(s)$ can be cost function of such solution *s*. The smaller the energy is, the less the costing price is. For example with optimization problem, $E(s)$ can be target function itself $E(\boldsymbol{x}) = f(\boldsymbol{x})$ in minimization and inverse of target function $E(\boldsymbol{x}) = -f(\boldsymbol{x})$ in maximization. However how to define exactly $E(s)$ depends on concrete applications.

How to define the neighbor function neighbor(*s*) is an interesting feature of SA because it is most flexible and depends on concrete applications. For example, given discrete states along with their transition probabilities, neighbor of state *s* can be a state *s*' so that transition probability from state *s* to state *s*' is maximum among other ones. Another example is the popular puzzle of travelling salesman problem in which state is a permutation of cities and hence, neighbor of a state *s* is obtained by swapping two arbitrary cities in *s*.

About combination of SA and PSO, Sait et al. (Sait, Sheikh, & El-Maleh, 2013) embedded SA into PSO by calling SA after some PSO iterations whose global best position is not improved. Their ideology (Sait, Sheikh, & El-Maleh, 2013, p. 656) can be explained as follows:

Repeat

Update velocities and positions of all particles according to PSO equations 1.1 and 1.2.

$$\boldsymbol{v}_i = \boldsymbol{v}_i + U(0, \phi_1) \otimes (\boldsymbol{p}_i - \boldsymbol{x}_i) + U(0, \phi_2) \otimes (\boldsymbol{p}_g - \boldsymbol{x}_i)$$

$$\boldsymbol{x}_i = \boldsymbol{x}_i + \boldsymbol{v}_i$$

Reset best positions $\boldsymbol{p}_i$ and $\boldsymbol{p}_g$ for all particles and for entire swarm.

If $\boldsymbol{p}_g$ is not improved after some *k* iterations, SA will be revoked to improve all particle positions with note that each position $\boldsymbol{x}_i$ is considered a state.

Until terminated conditions are met.

Table 3.2.4. Embedding SA into PSO

Note, the goal of the SA invoking is to improve every $\boldsymbol{x}_i$ and so, if there are *N* particles, SA will be called *N* times such that each time for one $\boldsymbol{x}_i$ with expectation that each $\boldsymbol{x}_i$ will reach a local optimizer. In other words, the goal of the SA invoking is to improve exploitation ability of PSO. Therefore, it is easy to deduce that SA is suitable for local optimization. The embedding SA into PSO is highly significant in case that the SA invoking process focuses on optimizing target function with minimizing some constraint (costing price) $g(\boldsymbol{x})$ so that SA will define the energy based on $g(\boldsymbol{x})$, for example $E(\boldsymbol{x}) = g(\boldsymbol{x})$.

4. Discussions

PSO is the simple but effective method to solve the optimization problem. Its ideology and practice are reasonable and interesting but there are not fully comprehensive mathematical models for PSO yet (Poli, Kennedy, & Blackwell, 2007, p. 14). However, it is necessary to survey some models to understand PSO and its preeminent features. There are two popular mathematical models for PSO such as deterministic model and stochastic process model. Deterministic model (DM) is simplest one because it fixes random numbers in surveying velocity update rule. Stochastic process model (SPM) considers movement of particles as stochastic process. Recall that the velocity update rule and the position update rule are:

$$\boldsymbol{v}_i = \omega \boldsymbol{v}_i + U(0, \phi_1) \otimes (\boldsymbol{p}_i - \boldsymbol{x}_i) + U(0, \phi_2) \otimes (\boldsymbol{p}_g - \boldsymbol{x}_i)$$
$$\boldsymbol{x}_i = \boldsymbol{x}_i + \boldsymbol{v}_i$$

When time points are considered explicitly, these rules are rewritten:

$$\boldsymbol{v}(t+1) = \omega \boldsymbol{v}(t) + U(0, \phi_1(t)) \otimes (\boldsymbol{p}(t) - \boldsymbol{x}(t)) + U(0, \phi_2(t)) \otimes (\boldsymbol{p}_g(t) - \boldsymbol{x}(t))$$
$$\boldsymbol{x}(t+1) = \boldsymbol{x}(t) + \boldsymbol{x}(t)$$

Without loss of generality, let $r_1(t)$ and $r_2(t)$ be two random variables which represent random-number-making functions $U(0, \phi_1)$ and $U(0, \phi_2)$, respectively, we have (Cleghorn & Engelbrecht, 2014, p. 38):

$$\begin{aligned} \boldsymbol{v}(t+1) &= \omega \boldsymbol{v}(t) + \phi_1 r_1(t) (\boldsymbol{p}(t) - \boldsymbol{x}(t)) + \phi_2 r_2(t) (\boldsymbol{p}_g(t) - \boldsymbol{x}(t)) \\ \boldsymbol{x}(t+1) &= \boldsymbol{x}(t) + \boldsymbol{v}(t+1) \end{aligned} \quad (4.1)$$

The three components such as inertial component, cognitive component, and social component are $\omega \boldsymbol{v}(t)$, $\phi_1 r_1(t)(\boldsymbol{p}(t) - \boldsymbol{x}(t))$, and $\phi_2 r_2(t)(\boldsymbol{p}_g(t) - \boldsymbol{x}(t))$, respectively. The main purpose of mathematical models is to prove the convergence of PSO. Suppose there is a stable point $\boldsymbol{x}^*$, a PSO process converges to $\boldsymbol{x}^*$ if (Cleghorn & Engelbrecht, 2014, p. 40)

$$\lim_{t \to +\infty} |\boldsymbol{x}(t) - \boldsymbol{x}^*| = 0 \quad (4.2)$$

Where |.| denotes the distance or norm in Eclidean space.

Deterministic model (DM), first was proposed by Ozcan and Mohan 1998, raises a so-called *deterministic assumption* that $\phi_1 r_1(t)$ and $\phi_2 r_2(t)$ are not dependent on time point t (Cleghorn & Engelbrecht, 2014, p. 40).

$$\begin{aligned} \theta_1 &= \phi_1 r_1(t) \\ \theta_2 &= \phi_2 r_2(t) \end{aligned} \quad (4.3)$$

The deterministic assumption is very strict, but it is necessary for DM to prove the convergence of PSO. DM also raises another assumption called *stagnation assumption* (Cleghorn & Engelbrecht, 2014, p. 41) in which local best position $\boldsymbol{p}(t)$ and global best position $\boldsymbol{p}_g(t)$ are not dependent on time point t and moreover, they are equal.

$$\boldsymbol{p}(t) = \boldsymbol{p}_g(t) = \boldsymbol{p} \quad (4.4)$$

For easy calculation, DM supposes that positions and velocities are scalars:

$$\boldsymbol{v}(t) = v(t)$$
$$\boldsymbol{x}(t) = x(t)$$
$$p(t) = p_g(t) = p$$

Hence, velocity update rule and position update rule in DM are:

$$v(t+1) = \omega v(t) + \theta_1 (p - x(t)) + \theta_2 (p - x(t))$$
$$x(t+1) = x(t) + v(t+1)$$

Without loss of information, let $\omega = 1$ and $\theta = \theta_1 + \theta_2$, we have (Cleghorn & Engelbrecht, 2014, p. 41):

$$\begin{aligned} v(t+1) &= v(t) + \theta (p - x(t)) \\ x(t+1) &= x(t) + v(t+1) \end{aligned} \quad (4.5)$$

Suppose we have initial velocity $v(0) = v_0$, initial position $x(0) = x_0$, it is easy to deduce the position $x(t)$ at time point t by recurrence formulation as follows (Cleghorn & Engelbrecht, 2014, p. 41):

$$x(t) = (\alpha + \beta)\left(1 - \frac{\Delta - \theta}{2}\right)^t + p \tag{4.6}$$

Where,

$$\Delta = \sqrt{\theta^2 - 4\theta}$$
$$\alpha = (x_0 - p - \beta)$$
$$\beta = 2(x_0 - p)(\Delta + \theta)\Delta - \frac{v_0}{\Delta}$$

If Δ is valid then $\Delta \geq \theta$ such that

$$\lim_{t \to +\infty} x(t) = \lim_{t \to +\infty} \left((\alpha + \beta)\left(1 - \frac{\Delta - \theta}{2}\right)^t + p \right) = p$$

As a result, the convergence of PSO is asserted by DM. There are some other researches (Cleghorn & Engelbrecht, 2014, pp. 41-44) which alleviate the stagnation assumption by not equal assigning $p(t) = p_g(t)$ but still assuming the time independence $p(t) = p$ and $p_g(t) = p_g$, which draw the same assertion about the convergence of PSO.

By more elegant approach, stochastic process model (SPM) considers the position variables $\boldsymbol{x}(t+1)$, $\boldsymbol{x}(t)$, and $\boldsymbol{x}(t-1)$ as random variables in stochastic process. From

$$\boldsymbol{v}(t+1) = \omega \boldsymbol{v}(t) + \phi_1 r_1(t)\big(\boldsymbol{p}(t) - \boldsymbol{x}(t)\big) + \phi_2 r_2(t)\left(\boldsymbol{p}_g(t) - \boldsymbol{x}(t)\right)$$
$$\boldsymbol{x}(t+1) = \boldsymbol{x}(t) + \boldsymbol{v}(t+1)$$

With SPM, Jiang and Yang (Jiang, Luo, & Yang, 2006, p. 10) deduces:

$$\boldsymbol{x}(t+1) = \Big(1 + \omega - \big(\phi_1 r_1(t) + \phi_2 r_2(t)\big)\Big)\boldsymbol{x}(t) - \omega \boldsymbol{x}(t-1) + \phi_1 r_1(t)\boldsymbol{p}(t) + \phi_2 r_2(t)\boldsymbol{p}_g(t)$$

Jiang and Yang assumed that local best position $\boldsymbol{p}(t)$ and global best position $\boldsymbol{p}_g(t)$ are not dependent on time points such that $\boldsymbol{p}(t) = \boldsymbol{p}$ and $\boldsymbol{p}_g(t) = \boldsymbol{p}_g$, which is the stagnation assumption. Therefore, they gave (Jiang, Luo, & Yang, 2006, p. 10):

$$\boldsymbol{x}(t+1) = \Big(1 + \omega - \big(\phi_1 r_1(t) + \phi_2 r_2(t)\big)\Big)\boldsymbol{x}(t) - \omega \boldsymbol{x}(t-1) + \phi_1 r_1(t)\boldsymbol{p} + \phi_2 r_2(t)\boldsymbol{p}_g \tag{4.7}$$

Because means of $r_1(t)$ and $r_2(t)$ are 0.5 when they are randomized in interval [0, 1], according to Jiang and Yang (Jiang, Luo, & Yang, 2006, p. 10), the expectation of $\boldsymbol{x}(t+1)$ is:

$$E\big(\boldsymbol{x}(t+1)\big) = \left(1 + \omega - \frac{\phi_1 + \phi_2}{2}\right)E\big(\boldsymbol{x}(t)\big) - \omega E\big(\boldsymbol{x}(t-1)\big) + \frac{\phi_1 \boldsymbol{p} + \phi_2 \boldsymbol{p}_g}{2} \tag{4.8}$$

The probability distribution under the expectation is the probability distribution of stochastic process. Jiang and Yang (Jiang, Luo, & Yang, 2006, p. 10) proved that given $\omega \geq 0$, $\phi_1 \geq 0$, and $\phi_2 \geq 0$, the stochastic process represented by $E(\boldsymbol{x}(t+1))$ or $E(\boldsymbol{x}(t))$ will converge to the point $(\phi_1 \boldsymbol{p} + \phi_2 \boldsymbol{p}_g)/2$ if and only if $0 \leq \omega < 1$ and $0 < \phi_1 + \phi_2 < 4(1 + \omega)$ (Jiang, Luo, & Yang, 2006, p. 10).

$$\lim_{t \to +\infty} E\big(\boldsymbol{x}(t+1)\big) = \frac{\phi_1 \boldsymbol{p} + \phi_2 \boldsymbol{p}_g}{2} \text{ if and only if } \begin{cases} 0 \leq \omega < 1 \\ 0 < \phi_1 + \phi_1 < 4(1 + \omega) \end{cases}$$

This result also fits with the recent study of minima study (Luo, 2019) which stated that the integral (expectation) of the target function $f(\boldsymbol{x})$ under a probability distribution of minimizer searching movement will converge to global optimal value $f(\boldsymbol{x}^*)$.

References

al-Rifaie, M. M., & Blackwell, T. (2012). Bare Bones Particle Swarms with Jumps. In M. Dorigo, M. Birattari, C. Blum, A. L. Christensen, A. P. Engelbrecht, R. Groß, & T.

Stützle (Ed.), *International Conference on Swarm Intelligence. Lecture Notes in Computer Science 7461*, pp. 49-60. Brussels: Springer Berlin. doi:10.1007/978-3-642-32650-9_5

Bonyadi, M. R., & Michalewicz, Z. (2017, March 2). Particle Swarm Optimization for Single Objective Continuous Space Problems: A Review. (N. Lindsay, Ed.) *Evolutionary Computation, 25*(1), 1-54. doi:10.1162/EVCO_r_00180

Cleghorn, C. W., & Engelbrecht, A. P. (2014, January 14). A Generalized Theoretical Deterministic Particle. *Swarm Intelligence, 8*, 35-39. doi:10.1007/s11721-013-0090-y

Fu, X., Liu, W., Zhang, B., & Deng, H. (2013, October 24). Quantum Behaved Particle Swarm Optimization with Neighborhood Search for Numerical Optimization. *Mathematical Problems in Engineering, 2013*. doi:10.1155/2013/469723

Guner, A. R., & Sevkli, M. (2008, April 8). A Discrete Particle Swarm Optimization Algorithm for Uncapacitated Facility Location Problem. *Journal of Artificial Evolution and Applications, 2008*, 1-9. doi:10.1155/2008/861512

Jiang, M., Luo, Y., & Yang, S. (2006, November 17). Stochastic convergence analysis and parameter selection of the standard particle swarm optimization algorithm. (A. Tarlecki, Ed.) *Information Processing Letters, 102*(1), 8-16. doi:10.1016/j.ipl.2006.10.005

Karaboga, D., & Akay, B. (2009, April 11). A comparative study of Artificial Bee Colony algorithm. *Applied Mathematics and Computation, 214*(1), 108-132. doi:10.1016/j.amc.2009.03.090

Luo, X. (2019, May 24). Minima distribution for global optimization. *arXiv preprint*. doi:10.48550/arXiv.1812.03457

Mallawaarachchi, V. (2017, July 8). *Introduction to Genetic Algorithms — Including Example Code*. (medium.com) Retrieved from Towards Data Science: https://towardsdatascience.com/introduction-to-genetic-algorithms-including-example-code-e396e98d8bf3

Pan, F., Hu, X., Eberhart, R., & Chen, Y. (2008, September 21). An Analysis of Bare Bones Particle Swarm. *IEEE Swarm Intelligence Symposium 2008 (SIS 2008)* (pp. 1-5). St. Louis, MO, US: IEEE. doi:10.1109/SIS.2008.4668301

Parsopoulos, K. E., & Vrahatis, M. N. (2002, January). Particle Swarm Optimization Method for Constrained Optimization Problems. In P. Sincak, P. Sincak, J. Vascak, V. Kvasnicka, & J. Pospichal (Eds.), *Intelligent Technologies from Theory to Applications: New Trends in Intelligent Technologies (Frontiers in Artificial Intelligence and Applications)* (Vol. 76, pp. 214-220). IOS Pres. Retrieved from https://www.cs.cinvestav.mx/~constraint/papers/eisci.pdf

Poli, R., Kennedy, J., & Blackwell, T. (2007, June). Particle swarm optimization. (M. Dorigo, Ed.) *Swarm Intelligence, 1*(1), 33-57. doi:10.1007/s11721-007-0002-0

Qiu, C., Wang, C., & Zuo, X. (2013, May 16). A novel multi-objective particle swarm optimization with K-means based global best selection strategy. *International Journal of Computational Intelligence Systems, 6*(5), 822-835. doi:10.1080/18756891.2013.805584

Ratnaweera, A., Halgamuge, S. K., & Watson, H. C. (2004, June 14). Self-organizing hierarchical particle swarm optimizer with time-varying acceleration coefficients. (R. C. Eberhart, & Y. Shi, Eds.) *IEEE Transactions on Evolutionary Computation, 8*(3), 240-255. doi:10.1109/TEVC.2004.826071

Sait, S. M., Sheikh, A. T., & El-Maleh, A. H. (2013, October). Cell assignment in hybrid CMOS/nanodevices architecture using a PSO/SA hybrid algorithm. *Journal of Applied Research and Technology, 11*(5), 653-664. doi:10.1016/S1665-6423(13)71573-6

Sharma, T. K., Pant, M., & Abraham, A. (2013). Blend of Local and Global Variant of PSO in ABC. *The 2013 World Congress on Nature and Biologically Inspired Computing.* IEEE. doi:10.1109/NaBIC.2013.6617848

Too, J., Abdullah, A. R., & Saad, N. M. (2019, May 8). A New Co-Evolution Binary Particle Swarm Optimization with Multiple Inertia Weight Strategy for Feature Selection. (A. Bryant, R. Theron, K. Sedig, & D. J. Lizotte, Eds.) *informatics, 6*(2), 1-14. doi:10.3390/informatics6020021

Wang, M., Wang, L., Xu, X., Qin, Y., & Qin, L. (2019, March 20). Genetic Algorithm-Based Particle Swarm Optimization Approach to Reschedule High-Speed Railway Timetables: A Case Study in China. (L. Dell'Olio, Ed.) *Journal of Advanced Transportation, 2019*, 1-11. doi:10.1155/2019/6090742

Wikipedia. (2002, February 25). *Genetic algorithm*. (Wikimedia Foundation) Retrieved from Wikipedia website: https://en.wikipedia.org/wiki/Genetic_algorithm

Wikipedia. (2003, January 21). *Simulated annealing*. (Wikimedia Foundation) Retrieved from Wikipedia website: https://en.wikipedia.org/wiki/Simulated_annealing

Wikipedia. (2005, November 11). *Differential evolution*. (Wikimedia Foundation) Retrieved from Wikipedia website: https://en.wikipedia.org/wiki/Differential_evolution

Wikipedia. (2007, November 23). *Artificial bee colony algorithm*. (Wikimedia Foundation) Retrieved from Wikipedia website: https://en.wikipedia.org/wiki/Artificial_bee_colony_algorithm

Wikipedia. (2017, March 7). *Particle swarm optimization*. (Wikimedia Foundation) Retrieved April 8, 2017, from Wikipedia website: https://en.wikipedia.org/wiki/Particle_swarm_optimization

Yu, H., Gao, Y., & Wang, J. (2020, December 1). A Multiobjective Particle Swarm Optimization Algorithm Based on Competition Mechanism and Gaussian Variation. (Z. Yang, Ed.) *Complexity, 2020*(si597427), 1-23. doi:10.1155/2020/5980504

Yu, X., Cao, J., Shan, H., Zhu, L., & Guo, J. (2014, February 9). An Adaptive Hybrid Algorithm Based on Particle Swarm Optimization and Differential Evolution for Global Optimization. (T. Chen, Q. Cheng, & J. Yang, Eds.) *The Scientific World Journal, 2014*(215472), 1-16. doi:10.1155/2014/215472

Zhang, Y., Wang, S., & Ji, G. (2015, October 7). A Comprehensive Survey on Particle Swarm Optimization Algorithm and Its Applications. (G. Xie, Ed.) *Mathematical Problems in Engineering, 2015*(Special Issue: Artificial Intelligence and Its Applications 2014), 1-38. doi:10.1155/2015/931256

Contents