Int. J. Advance. Soft Comput. Appl., Vol. 5, No. 1, March 2013
ISSN 2074-8523; Copyright © ICSRS Publication, 2013
www.i-csrs.org

A Review of Population-based Meta-Heuristic Algorithm

Zahra Beheshti, Siti Mariyam Hj. Shamsuddin

Soft Computing Research Group, Faculty of Computing, Universiti Teknologi Malaysia (UTM), Skudai, 81310 Johor, Malaysia
e-mail: bzahra2@live.utm.my

Soft Computing Research Group, Faculty of Computing, Universiti Teknologi Malaysia (UTM), Skudai, 81310 Johor, Malaysia
e-mail: mariyam@utm.my

Abstract

Exact optimization algorithms are not able to provide an appropriate solution in solving optimization problems with a high-dimensional search space. In these problems, the search space grows exponentially with the problem size therefore; exhaustive search is not practical. Also, classical approximate optimization methods like greedy-based algorithms make several assumptions to solve the problems. Sometimes, the validation of these assumptions is difficult in each problem. Hence, meta-heuristic algorithms which make few or no assumptions about a problem and can search very large spaces of candidate solutions have been extensively developed to solve optimization problems these days. Among these algorithms, population-based meta-heuristic algorithms are proper for global searches due to global exploration and local exploitation ability. In this paper, a survey on meta-heuristic algorithms is performed and several population-based meta-heuristics in continuous (real) and discrete (binary) search spaces are explained in details. This covers design, main algorithm, advantages and disadvantages of the algorithms.

Keywords: *Optimization, Meta-heuristic algorithm, Population-based meta-heuristic Algorithm, high dimension search space, Continuous and discrete search spaces.*

1 Introduction

In the past decades, many optimization algorithms, including exact and approximate algorithms, have been proposed to address optimization problems. In the class of exact optimization algorithms, the design and implementation of algorithms are usually based on methods such as dynamic programming, backtracking and branch-and-bound methods [1]. However, these algorithms have a good performance in many problems [1, 2, 3, 4, 5], they are not efficient in solving larger scale combinatorial and highly non-linear optimization problems. Due to the fact that the search space increases exponentially with the problem size and exhaustive search is impractical in these problems. Also, traditional approximate methods like greedy algorithms usually require making several assumptions which might not be easy to validate in many situations [1]. Therefore, a set of more adaptable and flexible algorithms are required to overcome these limitations. Based on this motivation, several algorithms usually inspired by natural phenomena have been proposed in the literature. Among them, some meta-heuristic search algorithms with population-based framework have shown satisfactory capabilities to handle high dimension optimization problems.

Artificial Immune System (AIS) [6], Genetic Algorithm (GA) [7], Ant Colony Optimization (ACO) [8], Particle Swarm Optimization (PSO) [9, 10], Stochastic Diffusion Search (SDS) [11], Artificial Bee Colony (ABC) [12], Intelligent Water Drops (IWD) [13], River Formation Dynamics (RFD) [14], Gravitational Search Algorithm (GSA) [5] and Charged System Search (CSS) [16] are in the class of such algorithms. These algorithms and their improved scheme have shown a good performance in the wide range of problems such as neural network training [17, 18], pattern recognition [19, 20], function optimization [21, 22], image processing [23, 24], data mining [25, 26], combinatorial optimization problems [27, 28] and so on.

This paper provides an overview of meta-heuristic algorithms followed by population-based meta-heuristics. The structure of this paper is organized as: the basic concepts of meta-heuristic algorithms and meta-heuristics classification are described in Section 2 and Section 3 respectively. A brief review of related works is presented in Section 4. In Section 5, several population-based meta-heuristic algorithms in real and binary search spaces are provided in details. Finally, a discussion and summary of this paper will be demonstrated in Section 6 and Section 7.

2 Concept of meta-heuristic

The words of “meta” and ‘‘heuristic” are Greek where, “meta” is “higher level” or “beyond” and heuristics means ‘‘to find”, ‘‘to know”, ‘‘to guide an investigation” or ‘‘to discover” [29]. Heuristics [30] are methods to find good (near-) optimal

solutions in a reasonable computational cost without guaranteeing feasibility or optimality. In other words, meta-heuristics are a set of intelligent strategies to enhance the efficiency of heuristic procedures. Laporte and Osman [31] defined a meta-heuristic as: "An iterative generation process which guides a subordinate heuristic by combining intelligently different concepts for exploring and exploiting the search space, learning strategies are used to structure information in order to find efficiently near-optimal solutions."

According to Voss et al. [32], a meta-heuristic is: "an iterative master process that guides and modifies the operations of subordinate heuristics to efficiently produce high-quality solutions. It may manipulate a complete (or incomplete) single solution or a collection of solutions per iteration. The subordinate heuristics may be high (or low) level procedures, or a simple local search, or just a construction method."

A majority of these algorithms has a stochastic behavior and mimics biological or physical processes. Different categories have been considered to classify meta-heuristic algorithms so far. In the next section, their classification will be described from different viewpoints.

3 Classification of meta-heuristic algorithms

Different ways based on the selected characteristics have been proposed to classify meta-heuristics as shown in Fig. 1 [33]. This section briefly summarizes the most important classes including nature-inspired against non-nature inspired, population-based against single point search, dynamic against static objective function, single neighborhood against various neighborhood structures, and memory usage against memory-less methods [33, 34, 35].

3.1 Nature-inspired against non-nature inspired

This class is based on the origin of algorithm. The majority of meta-heuristics are nature-inspired algorithms such as Ant colony Optimization (ACO), Particle Swarm Optimization (PSO) and Genetic Algorithms (GA). Also, some of them are non-nature-inspired algorithms like Iterated Local Search (ILS) [36] and Tabu Search (TS) [37].

3.2 Population-based against single point search

Meta-heuristics can also be classified according to the number of solutions used at the same time. Trajectory methods, as illustrated in Fig. 2, are the algorithms working based on a single solution at any time and encompass local search-based meta-heuristics such as TS, ILS and Variable Neighborhood Search (VNS) [38]. Population-based algorithms perform search with multiple initial points in a parallel style like swarm-based meta-heuristics.

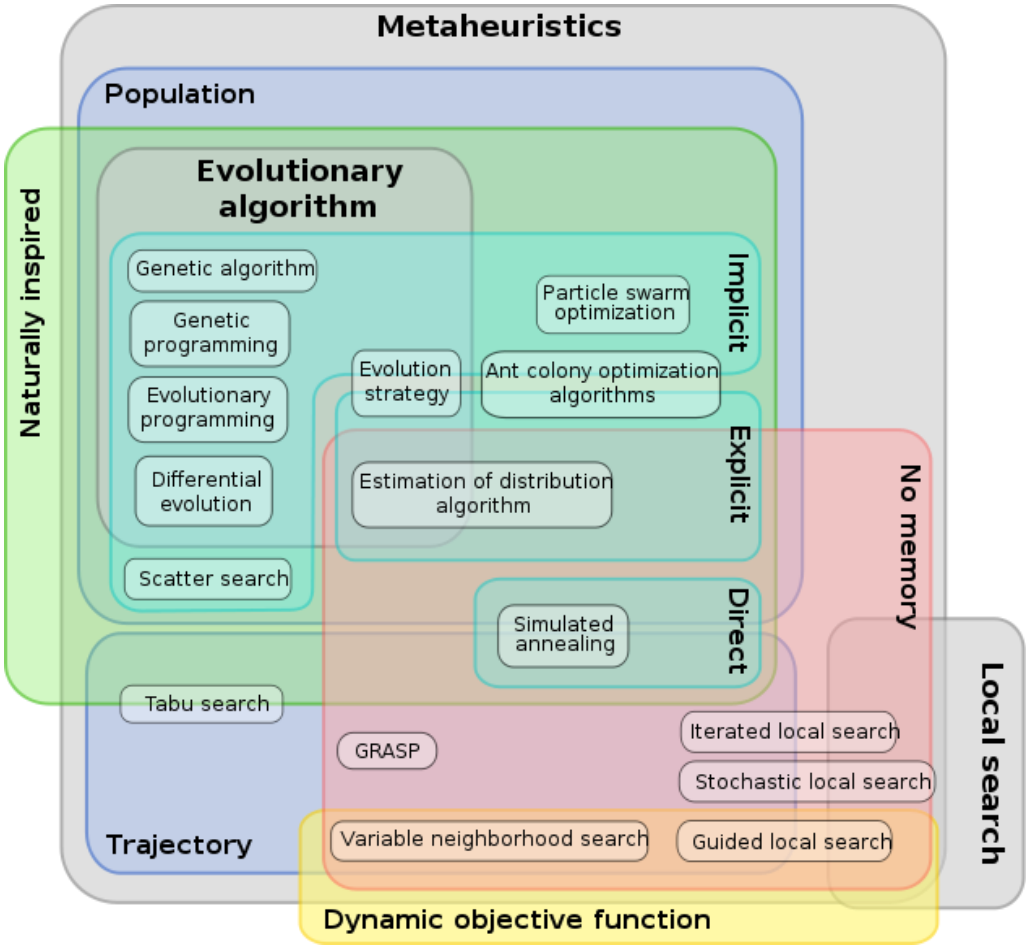

Fig. 1 Classification of meta-heuristic algorithms [33]

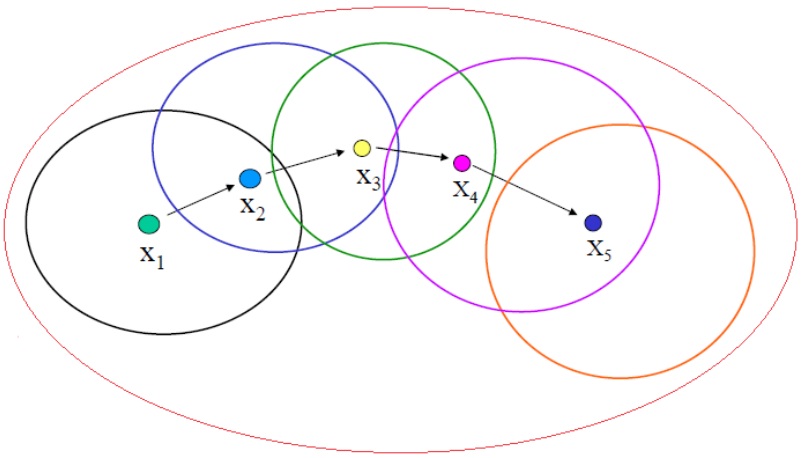

Fig. 2 Trajectory-based method

Swarm-based algorithms are made up of simple particles or agents interacting locally with each other and with their environment [39]. Each particle follows one or several rules without any centralized structure for controlling its behavior. Consequently, local and random interactions among the particles are led to an intelligent global behavior. These algorithms apply two approaches to achieve a proper performance [40]: global exploration and local exploitation.

The exploration is the potency of expanding search space, whereas the exploitation is the ability of finding the optimum solution around a good (near-) optimal solution. Due to insufficient knowledge about the search space in the initial steps, more exploration should be performed by the swarm to avoid trapping into local optima. In contrast, more exploitation is required by lapse of steps, so that the algorithm is able to tune itself in semi-optimal points. In other words, an appropriate trade-off between exploration and exploitation is necessary to have an efficient search.

To realize the concepts of exploration and exploitation, particles pass three phases inspired from nature in each step, namely, self-adaptation, cooperation and competition. In the self-adaptation phase, each particle enhances its performance. Particles collaborate together by transferring information in the cooperation phase and finally, they compete to survive in the competition phase. These concepts direct an algorithm toward finding the best solution. [37].

3.3 Dynamic against static objective function

Another characteristic which can be employed for the meta-heuristics classification is the way of utilizing the objective function. In other words, the objective function is kept "as it is" in the problem representation by some algorithms while some others, such as Guided Local Search (GLS) [41], modify it during the search. The idea behind this approach is to escape from local optima by changing the search landscape. Therefore, the objective function is modified by incorporating the collected information during the search process.

3.4 Various against single neighborhood structure

The majority of meta-heuristic algorithms apply one single neighborhood structure. In other words, the fitness landscape topology does not alter in the course of the algorithm while others, like Variable Neighborhood Search (VNS), employ a set of neighborhood structures. This latter structure gives the possibility to diversify the search by swapping between different fitness landscapes.

3.5 Memory usage against memory-less methods

The use of memory is one of the most important features to classify meta-heuristics. In other words, memory usage is known as one of the fundamental

elements of a powerful meta-heuristic. Memory-less algorithms carry out a Markov process, as the information which is used to determine the next action is the current state of the search process. There are several ways of using memory. Also, the use of short term usually is different from long term memory. The first usually keeps track of recently performed moves, visited solutions or, in general, decisions taken. The second is usually an accumulation of synthetic parameters about the search.

4 Related works

Many meta-heuristic algorithms have been proposed so far as shown in Table 1. Genetic Algorithm (GA) as a population-based meta-heuristic algorithm was suggested by Holland [42]. In the algorithm, a population of strings called chromosomes encodes candidate solutions for optimization problems.

Simulated Annealing (SA) is a local search meta-heuristic proposed by Kirkpatrick et al. [43] based on the way thermodynamic systems go from one energy level to another. In this method, each point x of the search space is a state and the function $f(x)$ is an internal energy of a physical system in that state. The goal is to transfer system from random initial state to a minimum energy state. Therefore, some neighboring states x' of the current state x are considered at each step and based on probabilities it is decided whether system stays in the state x or move to state x'. Finally, these probabilities move the system to states of lower energy until the system achieves a good state for the application.

Farmer et al. [6] introduced Artificial Immune System (AIS) simulated the structure and function of biological immune system to solve problems. The immune system defends the body against foreign or dangerous cells or substances which invade it. In fact, AIS copies the method which the human body acquires immunity using vaccination against diseases. The AIS applies the VACCINE-AIS algorithm for identifying the best solution from a given number of anti-bodies and antigens. In AIS, the decision points and solutions are anti-bodies and antigens in the immune system which are employed to solve optimization problems.

Tabu Search (TS) is a meta-heuristic local search algorithm introduced by Glover and McMillan [37] and formalized in [44, 45]. In local search, the near neighbors of each solution are checked in order to find an improved solution. In fact, TS applies a local search to move from the current solution x to an improved solution x' in the neighborhood of x. This process is continued until stopping criteria is satisfied.

Another population-based algorithm is Particle Swarm Optimization (PSO) offered by Kennedy and Eberhart [9]. It is a global optimization algorithm which the best solution can be represented as a point or surface in a multi-dimensional

search space. In the algorithm, particles are evaluated by their fitness values. They move towards those particles which have better fitness values and finally obtain the best solution.

Table 1: List of some meta-heuristic algorithms (1975-2012)

No.	Year	Algorithm
1.	1975	Holland introduced the Genetic Algorithm (GA).
2.	1977	Glover proposed Scatter Search (SS).
3.	1980	Smith elucidated genetic programming.
4.	1983	Kirkpatrick et al. proposed Simulated Annealing (SA).
5.	1986	Glover and McMillan offered Tabu Search (TS).
6.	1986	Farmer et al. suggested the Artificial immune system (AIS).
7.	1988	Koza registered his first patent on genetic programming.
8.	1989	Evolver provided the first optimization software using the GA.
9.	1989	Moscato presented Memetic Algorithm.
10.	1992	Dorigo proposed the Ant Colony Algorithm (ACO).
11.	1993	Fonseca and Fleming provided Multi-Objective GA (MOGA).
12.	1994	Battiti and Tecchiolli introduced Reactive Search Optimization (RSO) principles for the online self-tuning of heuristics.
13.	1995	Kennedy and Eberhart proposed Particle Swarm Optimization (PSO).
14.	1997	Storn and Price suggested Differential Evolution (DE).
15.	1997	Rubinstein presented the Cross Entropy Method (CEM).
16.	1999	Taillard and Voss proposed POPMUSIC.
17.	2001	Geem et al. provided Harmony Search (HS).
18.	2001	Hanseth and Aanestad offered Bootstrap Algorithm (BA).
19.	2004	Nakrani and Tovey presented Bees Optimization (BO).
20.	2005	Krishnanand and Ghose introduced Glowworm Swarm Optimization (GSO).
21.	2005	Karaboga proposed Artificial Bee Colony Algorithm (ABC).
22.	2006	Haddad et al. suggested Honey-bee Mating Optimization (HMO).
23.	2007	Hamed Shah-Hosseini offered Intelligent Water Drops (IWD).
24.	2007	Atashpaz-Gargari and Lucas introduced Imperialist Competitive Algorithm (ICA).
25.	2008	Yang presented Firefly Algorithm (FA).
26.	2008	Mucherino and Seref suggested Monkey Search (MS).
27.	2009	Husseinzadeh- Kashan provided League Championship Algorithm (LCA).
28.	2009	Rashedi et al. introduced Gravitational Search Algorithm (GSA)
29.	2009	Yang and Deb offered Cuckoo Search (CS).
30.	2010	Yang developed Bat Algorithm (BA).
31.	2011	Shah-Hosseini introduced the Galaxy-based Search Algorithm (GbSA).
32.	2011	Tamura and Yasuda designed Spiral Optimization (SO).
33.	2011	Rao et al. presented Teaching-Learning-Based Optimization (TLBO) algorithm.
34.	2012	Gandomi and Alavi proposed the Krill Herd (KH) Algorithm.
35.	2012	Çivicioglu introduced Differential Search Algorithm (DSA).

Ant Colony Optimization (ACO) [8] algorithm models the behavior of ants foraging and is useful for problems which require finding the shortest path as a goal. In real world, when ants explore their environment, it lays down the pheromones to direct each other toward resources. ACO also simulates this method and each ant records similarly its position so that more ants locate better solutions in later iterations. This trend continues until the best path is found.

Artificial Bee Colony (ABC) algorithm is another population-based presented by Karaboga [12]. ABC algorithm mimics the intelligent behavior of honey bees and applies three phases to find the best solution: employed bee, onlooker bee and scout bee phases. Employed and onlooker bees have local searches around the neighborhood and choose food based on the deterministic and probabilistic selection in their phases respectively. They select food sources based on their experience and their nest mates and modify their positions. In Scout phase, bees (scouts) fly and choose the food sources randomly without using experience. If the nectar amount of a new source is higher than that of the previous one in their memory, they memorize the new position and forget the previous one. Therefore, ABC balances exploration and exploitation process with local and global search methods in employed, onlooker and scouts phases and obtains the best solution.

Shah-Hosseini [13] proposed Intelligent Water Drops (IWD) algorithm inspired by the behavior of water drops in natural rivers. In the rivers, water drops find almost optimum paths to their destination through actions and reactions occurring among them and their riverbeds. The IWD algorithm simulates this trend and uses the constructive approach to find the optimum solution in a problem. Each artificial water drop constructs a solution (path) by traversing in the search space of the problem and modifying its environment. Among all these paths, the optimum or near optimum path is chosen by the algorithm.

Atashpaz-Gargari and Lucas [46] introduced Imperialist Competitive Algorithm (ICA). It is a socio-politically global search strategy to address different optimization problems. The algorithm achieves the best solution based on a competition among empires. The best solution is an imperialist with the best power.

Another population-based algorithm is River Formation Dynamics (RFD) [14] which is nature-inspired optimization and can be considered as a gradient version of ACO. The algorithm copies the behavior of water to form rivers. Waters shape the rivers by eroding the ground and depositing sediments. The altitudes of places are dynamically changed and decreasing gradients are created by transforming water in the environment. The gradients are followed by subsequent drops to make new gradients, emphasizing the best ones. By this method, good solutions are obtained in the form of decreasing altitudes. Due to the gradient orientation of RFD, the algorithm is proper to solve problems which use graphs and trees.

Gravitational Search Algorithm (GSA) was proposed based on the Newtonian gravity law and mass interactions by Rashedi et al. [15]. In the algorithm, objects in real world are considered as agents and their performance is measured by their masses which depend on fitness function values. The position of each agent in the search space shows a problem solution. The heaviest mass presents the optimum solution in the search space. By lapse of time, masses are attracted by the heaviest mass and are converged the best solution.

Also, Charged System Search (CSS) [16] was developed based on the Newtonian laws from mechanics and the governing Coulomb and Gauss laws from electrostatics. The algorithm is multi-agent and each agent is called Charged Particle (CP). Each CP has an electric force and affects other CPs according to the Coulomb and Gauss laws. This force gives acceleration to CP and the velocity of each CP changes with time. The motion laws and the resultant forces determine the new position (new solution) of the CPs. These positions are evaluated and replaced with previous ones if these positions are better. The trend continues until the maximum number of iterations achieves and obtains the best result to that extent.

As the scope of this study is meta-heuristic population-based algorithms, in the next section, some of the algorithms are explained in details for real and binary search spaces.

5 Population-based meta-heuristic algorithms

This section describes GA and PSO as older, GSA and ICA as newer population-based algorithms in continuous and discrete search spaces although ICA has been defined only on real search space.

5.1 Population-based meta-heuristic algorithms in real search space

The majority of meta-heuristic algorithms have been designed for real valued vectors in search spaces. In this section, GA, PSO, ICA and GSA are elucidated in the real search space, although GA is applied for both real and binary search space.

5.1.1 Genetic Algorithm (GA)

GA simulates the biological evolution process of chromosomes using *selection*, *crossover* and *mutation* operators. Chromosomes present problem solutions and are evaluated based on their fitness function to select parents. *Selection* is an important process for choosing parents to reproduce new population and can affect the convergence of GA. The convergence speed of different selection schemes was first studied by Goldberg and Deb [47]. The commonly selection

methods are proportionate reproduction, tournament selection, ranking selection and Genitor or steady state selection.

Roulette Wheel Selection or stochastic sampling with replacement is the simplest method in proportionate reproduction. In this technique, parents are selected based on their fitness. It means that chromosomes with better fitness have more chance to be chosen as demonstrated in Fig. 3. For each chromosome *i*, a probability is calculated as:

$$p_i = \frac{f_i}{\sum_{j=1}^{N} f_j}, \tag{1}$$

Where, f_i represents the fitness of chromosome *i* and *N* is population size.

Then, an ascending array is made according to the probabilities of the chromosomes. *N* random numbers are generated in the range 0 to $\sum_{j=1}^{N} f_j$ and chromosomes are chosen based on their probabilities of selection. As demonstrated in Fig. 3, $Chromosome_1$ has more chance than others to be selected. However, when the difference of the fitness values is very much, the method is inefficient. For instance, if the best chromosome fitness is 95% of the roulette wheel, the other chromosomes will have a few chances to be chosen.

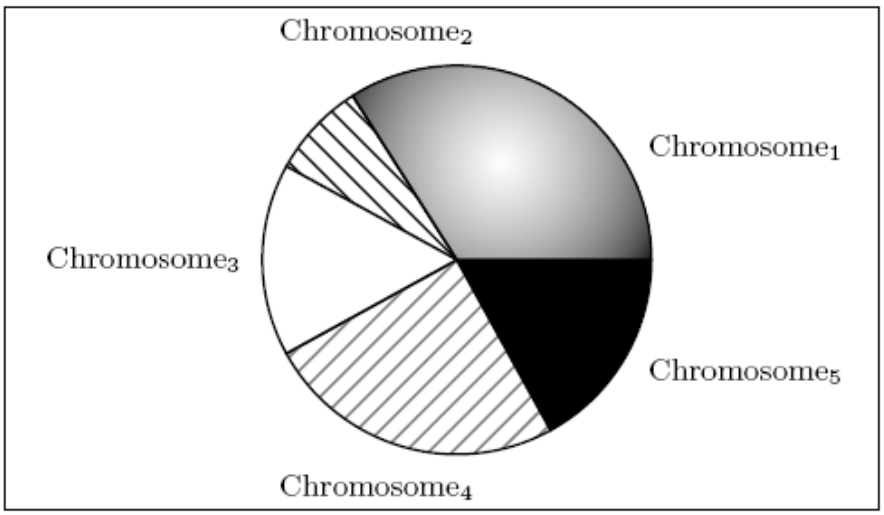

Fig. 3 Roulette wheel selections

In *Tournament Selection*, a number of chromosomes are randomly selected from the population and the two best are chosen as parents. The parents produce offspring and this process is often repeated until reaching a tournament size. The tournament size usually depends on the population size and takes values ranging from 2 to population size.

In the method of *ranking selection*, each chromosome has a rank in population. It means that the worst fitness will have fitness 1and the best will have fitness N (population size). After this, all the chromosomes have a chance to be chosen. In this method the best chromosomes are not considerably different from others hence, it leads to slow convergence rate.

The main idea of *Genitor or steady state selection* is to survive the big part of chromosomes in next generation. Therefore, a few chromosomes with high fitness are chosen for creating a new offspring in each generation. Then, some chromosomes with low fitness are deleted and the new offspring is replaced in their place. The rest of population survives to generate new generation.

Other operators in GA are *Crossover* and *Mutation.* In *Crossover*, some parts of two chromosomes are exchanged together. The crossover performs using several methods such as one, two and uniform crossover as demonstrated in Fig. 4. Therefore, one, two and several parts of parents' chromosomes are exchanged in one, two and uniform crossover respectively.

In *Mutation*, some parts of chromosomes randomly change in order to have better performance and escape from local optima. It is possible to lose the best chromosomes when new chromosomes are created by crossover and mutation operators.

Elitism is a method which copies the best chromosome (or a few best chromosomes) to new population and the rest is created by the mentioned methods. Therefore, this method increases the performance of GA, because it avoids losing the best-found solution [48]. The above steps have been illustrated in Fig. 5.

Although GA has been extensively used in various problems, it suffers from some disadvantages [49, 50, 51] such as:

1. Using complex operators for selection and cross over.
2. Unpredicted results.
3. Premature convergence rate.
4. Trapping into local optima.
5. Taking long run-time.
6. Weak local search.

7. .Difficult encoding scheme

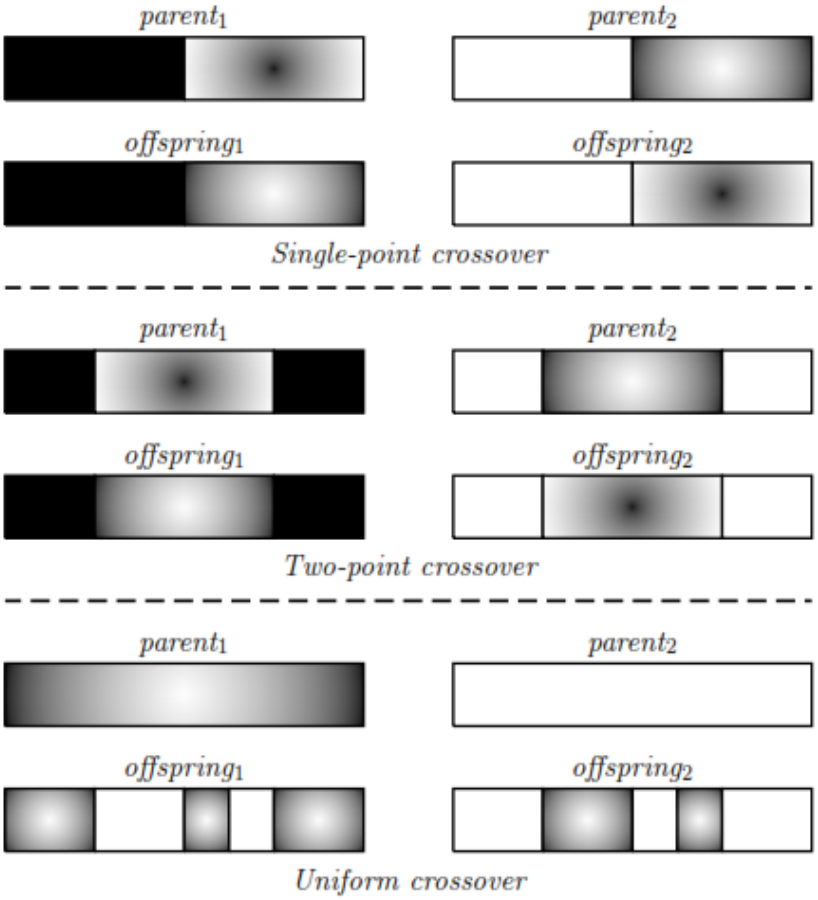

Fig. 4 GA crossover

Step 1: *Start.*
Step 2: *Create first generation of chromosomes.*
Step 3: *Define Parameters and fitness function.*
Step 4: *Calculate the fitness of each individual chromosome.*
Step 5: *Choose the chromosomes by Elitism method.*
Step 6: *Select a pair of chromosomes as parents.*
Step 7: *Perform Crossover and Mutation to generate new chromosomes.*
Step 8: *Combine the new chromosomes and the chromosomes of Elitism Set in the new population (the next generation).*
Step 9: *Repeat Step 4 to Step 8 until reaching termination criteria.*
Step 10: *Return best solution.*

Fig. 5 GA pseudo code

5.1.2 Particle Swarm Optimization (PSO) in real search space

PSO is a swarm-based meta-heuristic algorithm which simulates the flock of birds or insects motion in order to find the best solution. In the algorithm, the birds or insects called particles and are initialized by random positions and velocities [9, 10]. Each particle is described by a group of vectors denoted as ($\vec{X}_i, \vec{V}_i, \vec{P}_i$) in a *d–dimensional* search space, where, $\vec{X}_i$ and $\vec{V}_i$ are the position and velocity of the *ith* particle defined as:

$$\vec{X}_i = (x_{i1}, x_{i2}, ..., x_{id}) \quad for \quad i = 1,2,...,N . \tag{2}$$

$$\vec{V}_i = (v_{i1}, v_{i2}, ..., v_{id}) \quad for \quad i = 1,2,...,N . \tag{3}$$

$\vec{P}_i$ is the personal best position found by the *ith* particle:

$$\vec{P}_i = (p_{i1}, p_{i2}, ..., p_{id}) \quad for \quad i = 1,2,...,N . \tag{4}$$

Also, the best position achieved by the entire population ($\vec{P}_g$) is computed to update the particle velocity:

$$\vec{P}g = (p_{g1}, p_{g2}, ..., p_{gd}). \tag{5}$$

From $\vec{P}_i$ and $\vec{P}_g$, the next velocity and position of *ith* particle are updated by Eq. (6) and Eq. (7):

$$v_{id}(t+1) = w(t) \times v_{id}(t) + C_1 \times rand \times (p_{id}(t) - x_{id}(t)) + C_2 \times rand \times (p_{gd}(t) - x_{id}(t)), \tag{6}$$

$$x_{id}(t+1) = x_{id}(t) + v_{id}(t+1), \tag{7}$$

Where, v_{id} *(t+1)* and v_{id} *(t)* are the next and current velocity of *ith* particle respectively. *w* is inertia weight, C_1 and C_2 are acceleration coefficients, *rand* is uniformly random number in the interval of [0, 1] and *N* is the number of particles. x_{id} *(t+1)* and x_{id} *(t)* show the next and current position of *ith* particle.

In Eq. (6), the second and the third term are called cognition and social term respectively. Also, $|v_{id}| < v_{max}$ is considered and v_{max} is set to a constant based

on the bounds of solution space by users. A larger value of *w* encourages a global exploration (searching new areas) while a smaller inertia weight facilitates a local exploitation [52]. It is usually decreased from 0.9 to 0.4 [10, 52].

In PSO algorithm, two models for choosing $\vec{P_g}$ are considered known as *gbest* (or global topology) and *lbest* (or local topology) models. In global model, the position of each particle is influenced by the best-fitness particle of entire population in the search space whereas in the local model, each particle is affected by the best-fitness particle chosen from its neighborhood. According to Bratton and Kennedy [53], the *lbest* model can return better results than the *gbest* model in many problems however; it might have lower convergence rate than *gbest* model. The steps of PSO are shown in the Fig. 6.

Although PSO is easy to implement, it may face up to the slow convergence rate, parameter selection problem and easily get trapped in a local optimum due to its poor exploration when solving complex multimodal problems [54, 55, 56]. If a particle falls into a local optimum, sometimes it cannot get rid of itself from the position. In other words, if $\vec{P_g}$ obtained by the population is a local optimum and the current position and the personal best position of particle *i* are in the local optimum, the second and third term of Eq. (6) tend toward zero, also *w* is linearly decreasing to near zero. Consequently, the next velocity of particle *i* tends toward zero and its next position in Eq. (7) cannot change and the particle remains in the local optimum. Therefore, variant PSO algorithms have been proposed to improve the performance of PSO and to overcome these limitations.

Step 1: *Start.*
Step 2: *Initialize the velocities and positions of population randomly.*
Step 3: *Evaluate fitness values of particles.*
Step 4: *Update* $\boldsymbol{P_i}$ *if particle fitness value* $_i$ *is better than particle best fitness value* $_i$ *, for i = 1, .. ., N.*
Step 5: *Update* $\boldsymbol{P_g}$ *if particle fitness value* $_i$ *is better than global best fitness value, for i = 1, . . ., N.*
Step 6: *Update the next velocity of particles.*
Step 7: *Update the next position of particles.*
Step 8: *Repeat steps 3 to 7 until the stop criterion is reached.*
Step 9: *Return best solution.*

Fig. 6 PSO pseudo code

PSO is one of the most popular optimizer which has been widely applied in solving optimization problem. Hence, the enhancement of performance and theoretical studies of the algorithm have become attractive. Convergence analysis and stability studies have been reported in [57, 58, 59, 60, 61]. Also, some research on the performance of PSO has been done in terms of topological structures, parameter studies and combination with auxiliary operations [62, 63].

In topological structures, Kennedy and Mendes proposed a ring topological structure PSO (LPSO) [64] and a Von Neumann topological structure PSO (VPSO) [65] to enhance the performance in solving multimodal problems. Also, Dynamic Multi-Swarm PSO (DMS-PSO) introduced by Liang and Suganthan [66] to improve the topological structure in dynamic way. This new neighborhood structure has two important characters: Small sized swarms and randomly regrouping schedule. Since the small sized swarms are searching using their own best historical information, they are easy to converge to a local optimum because of PSO's convergence property. In this case, if the neighborhood structures are kept unchanged, no information is exchanged among the swarms. To avoid this situation, a randomized regrouping schedule is considered and the good information obtained by each swarm is exchanged among the swarms. The new neighborhood structure has better performance on complex multimodal problems than classical neighborhood structure. Other topology structures have been also proposed to improve the performance of PSO. For example, Fully Informed particle swarm (FIPS) algorithm [67] used the information of the entire neighborhood to influence the flying velocity.

In parameter studies, much research has been done on inertia weight w and acceleration coefficients C_1 and C_2. Shi and Eberhart [10, 52] suggested that the inertia weight w in Eq. (6) is linearly decreased by the iterative generations as Eq. (8):

$$w = w_{\max} - (w_{\max} - w_{\min}) \frac{iter}{Maxiter} \tag{8}$$

Where, *iter* is the current generation and *Maxiter* is maximum generations. The w_{max} and w_{min} are usually set to 0.9 and 0.4 respectively. Moreover, other values of w have been proposed to improve the searching ability of PSO. A fuzzy adaptive w was introduced and a random version setting w to $0.5 + random(0,1)/2$ was experimented for dynamic system optimization [68, 69]. Also, a constriction factor [57] was introduced based on Eq. (9) and the next velocity was computed according to Eq. (11):

$$\chi = \frac{2}{\left|2-\varphi-\sqrt{\varphi^2-4\varphi}\right|}, \tag{9}$$

$$\varphi = C_1 + C_2 = 4.1, \tag{10}$$

$$v_{id}(t+1) = \chi\left[v_{id}(t) + C_1 \times rand \times \left(p_{id}(t) - x_{id}(t)\right) + C_2 \times rand \times \left(p_{gd}(t) - x_{id}(t)\right)\right], \tag{11}$$

Where, C_1 and C_2 are both set to 2.05 and χ is mathematically equivalent to *w*, as Eberhart and Shi [70] pointed out.

The experiment results have illustrated that both acceleration coefficients C_1 and C_2 are essential to the success of PSO. Kennedy and Eberhart [9] offered a fixed value of 2.0, and this configuration has been adopted by many other researchers. Ratnaweera et al. [71] proposed self-organizing Hierarchical Particle Swarm Optimizer with Time-Varying Acceleration Coefficients (HPSO-TVAC) algorithm which used linearly time-varying acceleration coefficients, where a larger C_1 and a smaller C_2 were set at the beginning and were gradually reversed during the search. Therefore, particles allow moving around the search space instead of moving toward the population best at the beginning. The *w* in HPSO-TVAC is used as Eq. (8) and C_1 and C_2 are computed as:

$$C_j = (C_{jf} - C_{ji})\frac{iter}{Maxiter} + C_{ji}, \quad j = 1,2 \tag{12}$$

Where, C_{jf} and C_{ji} are the final and initial values of acceleration coefficients which are changed from 2.5 to 0.5 for C_1 and .from 0.5 to 2.5 for C_2.

Comprehensive Learning PSO (CLPSO) [54] is another PSO algorithm to improve the diversity of the swarm by encouraging each particle to learn from different particles on different dimensions. In the algorithm, each particle velocity can be updated by personal best position and other particles' best position. Hence, the velocity of each particle in CLPSO is given by:

$$v_{id}(t+1) = w(t) \times v_{id}(t) + C \times rand \times \left(p_{f_i(d),d}(t) - x_{id}(t)\right), \tag{13}$$

Where, $f_i = (f_{i1}, f_{i2}, ..., f_{id})$ *for* $i = 1,2,...,N$ defines which particles' best position the particle *i* should follow. $p_{f_i(d),d}$ can be the corresponding dimension of any particle's best position including its own best position and the decision depends on probability *Pc*, referred to as the learning probability, which can take

different values for different particles. For each dimension of particle *i*, a random number is generated. Hence, a tournament selection procedure has been suggested to choose randomly two particles and then select one with the best fitness as the exemplar to learn from for that dimension. CLPSO has only one acceleration coefficient *C* which is normally set to 1.494 and the inertia weight value is changed from 0.9 to 0.4.

An Adaptive PSO (APSO) was proposed by Zhan et al. [55]. In this algorithm, an evolutionary factor *f* is defined and computed with a fuzzy classification method to design effective parameter and to improve the speed of solving optimization problems. Hence, *w* changes based on a sigmoid mapping *w(f)* as shown in Eq. (14). The large *f* will benefit for the global search and convergence state is detected by small *f*.

$$w(f)=\frac{1}{1+1.5e^{-2.6f}}\in[0.4,0.9]. \qquad \forall f\in[0,1] \tag{14}$$

Moreover, acceleration coefficients are modified by increasing C_1 and decreasing C_2 in where the maximum increment or decrement between two generations is bounded by:

$$|C_i(t+1)-C_i(t)|\le\delta \qquad i=1,2 \qquad , \tag{15}$$

Where, δ is termed acceleration rate in interval [3.0, 4.0]. If the sum of C_1 and C_2 is larger than 4.0, then both C_1 and C_2 are normalized to:

$$C_i=\frac{C_i}{C_1+C_2}+4.0, \qquad i=1,2 \tag{16}$$

Another term in APSO is an Elitist Learning Strategy (ELS) to help $\vec{P_g}$ for jumping out of local optimal regions when the search is identified to be in a convergence state. If another better region is found for $\vec{P_g}$, then the rest of the swarm will follow to jump out and converge to the new region.

In addition to the mentioned algorithms, another active research trend in PSO is hybrid PSO with other evolutionary paradigms. Angeline [72] introduced a selection operation for PSO similar to GA. Also, hybridization of GA and PSO has been applied [73] for recurrent artificial neural network design.

Zhan et al. [74] introduced Orthogonal Learning Particle Swarm Optimization algorithm (OLPSO). The OL strategy could guide particles to discover useful information from the personal best position and its neighborhood's best position in order to fly in better directions. In another study, Gao et al. [56] used PSO with chaotic opposition-based population initialization and stochastic search technique to solve complex multimodal problems. The algorithm called CSPSO found new solutions in the neighborhoods of the previous best positions in order to escape from local optima in multimodal functions.

In other studies, Beheshti et al. proposed Median-oriented Particle Swarm Optimization (MPSO) [21] and Centripetal Accelerated Particle Swarm Optimization (CAPSO) [22] based on the improved scheme of PSO and Newtonian's motion laws. The algorithms do not require any specific-algorithm parameters and have been introduced for both local and global topology. Also, CAPSO has been proposed for both real and binary search spaces.

Although many extended PSO algorithms have been presented so far, the performance enhancement of PSO is an open problem because of its simple structure of PSO and easy to use.

5.1.3 Imperialist Competitive Algorithm (ICA)

ICA is a global search optimization algorithm inspired by imperialistic competition [46]. The algorithm is population-based including countries and imperialists. Each individual of the population is called a country. Some of the best countries with the best cost are chosen as imperialist states and others named colonies are divided among the imperialists according to their costs. Empires are formed by imperialist states and their colonies. After forming the empires, imperialistic competition is started among them to collapse weak empires and to remain the most powerful empire.

To divide the colonies among the imperialists, the cost and power of each imperialist are normalized and initial colonies are allocated to the empires as Eq. (17) to Eq. (19):

$$C_n = c_n - \max_i \{c_i\}, \tag{17}$$

$$p_n = \left| \frac{C_n}{\sum_{i=1}^{N_{imp}} C_i} \right|, \tag{18}$$

$$N.C_n = round\{p_n . N_{col}\}, \tag{19}$$

Where, c_n is the cost of the *nth* imperialist and C_n and p_n are the normalized cost and power of the imperialist respectively. Also, $N.C_n$ is the initial number of colonies related to the *nth* empire and N_{col} is the total number of initial colonies.
To form the *nth* empire, the $N.C_n$ of the colonies are randomly selected and allocated to the *nth* imperialist. In real world, the imperialist states try to make their colonies as part of themselves. This process, called assimilation, is modelled by moving all of the colonies toward the imperialist as illustrated in the Fig. 7. In this Figure, θ and x are random angle and number with uniform distribution also; d is the distance between the imperialist and colony.

$$x \sim U(0, \beta \times d), \tag{20}$$

$$\theta \sim U(-\gamma, \gamma), \tag{21}$$

Where, β , γ are parameters which cause colonies to move their relevant imperialist in a randomly deviated direction.

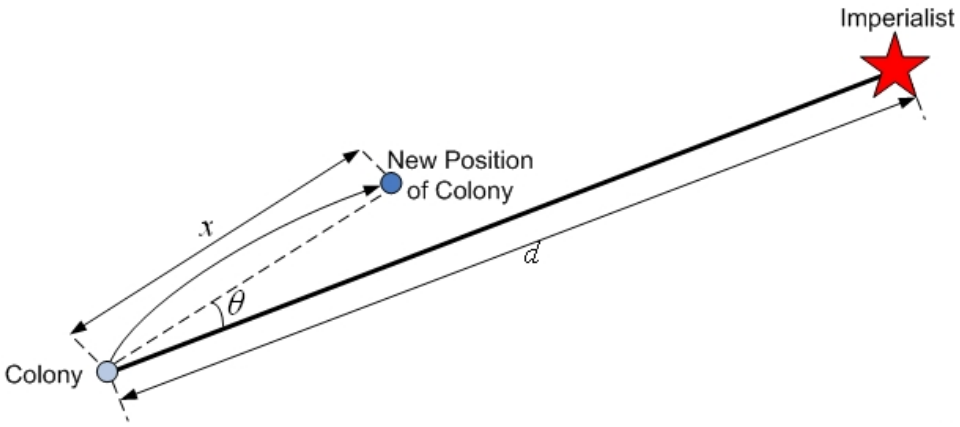

Fig. 7 Movement of colonies toward their relevant imperialist

While a colony moves toward an imperialist, it might achieve a better position than the imperialist. In this case, the colony and the imperialist change their positions with each other. Also, it is possible that a revolution takes place among colonies which changes the power or organizational structures of empire. In ICA, revolution operator alters the colony position and revolution rate shows the percentage of colonies which will randomly change their position.

Finally, empires compete together in order to possess and control other empires' colonies as shown in Fig. 8. A colony of the weakest empire is selected and the possession probability of each empire, *Pp*, is computed as Eq. (23). The

normalized total cost of an empire, $N.T.C_{n,}$ is acquired by Eq. (22) and used to obtain the empire possession probability.

$$N.T.C_n = T.C_n - \max_i \{T.C_i\}. \tag{22}$$

$$p_{p_n} = \left| \frac{N.T.C_n}{\sum_{i=1}^{N_{imp}} N.T.C_i} \right|. \tag{23}$$

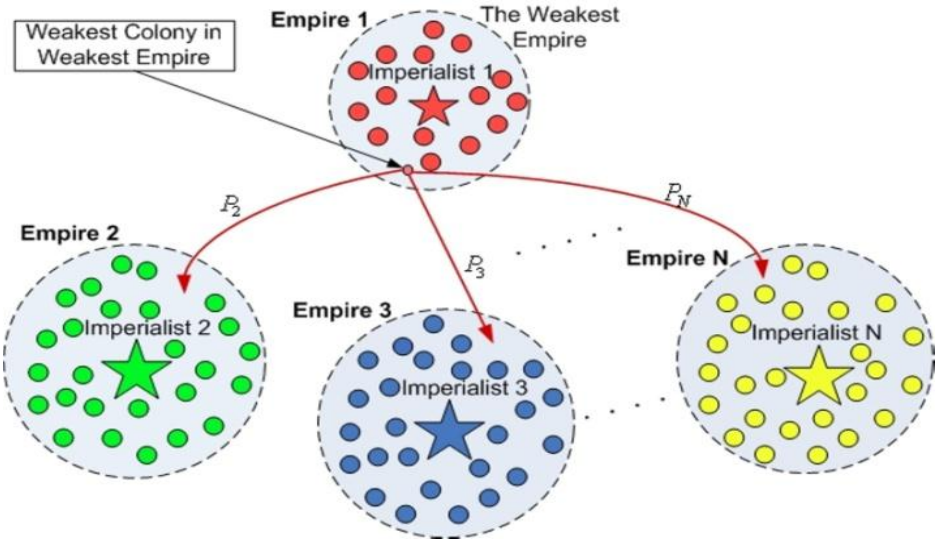

Fig. 8 Imperialistic competition

Vector P is formed in order to divide the mentioned colonies among empires:

$$P = \left[p_{p_1}, p_{p_2}, p_{p_3}, ..., p_{p_{N_{imp}}} \right]. \tag{24}$$

Vector R with the same size as P whose elements are uniformly distributed random numbers is created:

$$R = \left[r_1, r_2, r_3, ..., r_{N_{imp}} \right]$$
$$r_1, r_2, r_3, ..., r_{N_{imp}} \in U(0,1) \tag{25}$$

Vector *D* is formed so that the mentioned colony (colonies) is given to an empire whose relevant index in *D* is maximized.

$$D = P - R = \left[D_1, D_2, D_3, ..., D_{N_{imp}}\right]. \quad (26)$$

According to vector *D*, the process of choosing an empire is like the roulette wheel in GA. However, this method is faster because the selection is based on probabilities values.

The competition affects the power of empires and an empire power will be weaker or stronger. Therefore, all colonies of the weak empire are owned by more powerful empires and the weaker one is eliminated. The total power (cost) of an empire is modelled by adding the power of imperialist country (cost) and a percentage of mean power of its colonies (colonies costs) as follows:

$$T.C_n = Cost\left(imprialist_n\right) + \xi \ mean\left\{Cost\left(colonies \quad of \quad empire_n\right)\right\}, \quad (27)$$

Where, $T.C_n$ is the total cost of the *nth* empire and ξ is a positive small number.

The competition will be continued until remaining one empire or reaching determined maximum iteration. The above steps have been summarized in Fig. 9.

Step 1: *Start.*
Step 2: *Select some random points on the function and initialize the empires.*
Step 3: *Move the colonies toward their relevant imperialist (Assimilation).*
Step 4: *Randomly change the position of some colonies (Revolution).*
Step 5: *If there is a colony in an empire which has lower cost than the imperialist, exchange the positions of that colony and the imperialist.*
Step 6: *Unite the similar empires.*
Step 7: *Compute the total cost of all empires.*
Step 8: *Pick the weakest colony (colonies) from the weakest empires and give it (them) to one of the empires (Imperialistic competition).*
Step 9: *Eliminate the powerless empires.*
Step 10: *If stop conditions satisfied, stop, if not go to Step 3.*

Fig. 9 ICA pseudo code

Although ICA has shown good performance in many problems [75, 76, 77], it faces some drawbacks which cause the use of the algorithm to be difficult:

1. Using many equations and complex operators.
2. Long computational time.
3. Tuning many parameters.
4. Design only for continuous (real) search space.

5.1.4 Gravitational Search Algorithm (GSA) in real search space

GSA [15] is a meta-heuristic algorithm based on the Newtonian gravity and motion laws. According to the gravity law, objects attract each other by gravity force [78]. This force depends directly on the product of both objects masses and inversely proportional to the square of the distance between them. In GSA, the objects of real world are considered as agents and their masses depend on fitness function values. The position of each agent in the search space shows a problem solution. The heaviest mass presents an optimum solution in the search space. By lapse of time, masses are attracted by the heaviest mass and are converged to the best solution. Based on this law, GSA defines a system with *N* agents in a *d-dimensional* search space. The position of the *ith* agent is shown as:

$$\vec{X}_i = \left(x_{i1}, x_{i2}, ..., x_{id}\right) \quad for \quad i = 1,2,...,N \,. \tag{28}$$

Where, x_{id} presents the position of *ith* agent in the *dth* dimension.

At the beginning, these positions are initialized randomly. Then, the gravity force of mass *j* on mass *i* at specific time *t* is computed as follows:

$$F_{ij,d}(t) = G(t)\frac{M_i(t) \times M_j(t)}{R_{ij}(t) + \varepsilon}\left(x_{jd}(t) - x_{id}(t)\right), \tag{29}$$

Where, M_i and M_j are the masses of agent *i* and agent *j* respectively. ε is a small constant. R_{ij} (t) is the Euclidean distance between probe *i* and *j* at time *t*:

$$R_{ij}(t) = \left\| x_i(t), x_j(t) \right\|_2 . \tag{30}$$

Also, *G(t)* is gravitational constant initialized at the beginning and will be reduced with time *t* to control the search accuracy. *G* is a function with the initial value of G_0:

$$G(t) = G(G_0, t). \tag{31}$$

In the Eq. (29), M_i *(t)* is calculated as Eq. (33).

$$m_i(t) = \frac{fit_i(t) - worst(t)}{best(t) - worst(t)}, \tag{32}$$

$$M_i(t) = \frac{m_i(t)}{\sum_{j=1}^{N} m_j(t)}, \tag{33}$$

Where, $fit_i(t)$ is the fitness value of agent i at time t. Also, $best(t)$ and $worst(t)$ are the best and the worst values of fitness functions at time t.

The total force acting on agent i at time t in dimension d is considered as:

$$F_{id}(t) = \sum_{j \in kbest,\, j \neq i}^{N} rand_j \times F_{ij,d}(t), \tag{34}$$

Where, $rand_j$ is a random number in the range of [0, 1]. *Kbest* is the set of first K agents with the best fitness value and biggest mass. At the beginning, *Kbest* is initialized by K_0 and linearly reduced during the running time of algorithm.

Regarding the motion law, the force gives acceleration to the agent i:

$$a_{id}(t) = \frac{F_{id}(t)}{M_i(t)}. \tag{35}$$

This acceleration moves the agent from a position to another position. Hence, the next velocity of agent i in dimension d is computed as the sum of its current velocity and its acceleration:

$$v_{id}(t+1) = v_{id}(t) + rand_i \times a_{id}(t). \tag{36}$$

Also, the next position is considered as Eq. (37):

$$x_{id}(t+1) = x_{id}(t) + v_{id}(t+1), \tag{37}$$

Where, $v_{id}(t+1)$ and $x_{id}(t)$ are the next velocity and the current position of agent i in dimension d.

Fig. 10 shows the pseudo code of GSA. As seen, algorithm is initialized randomly and each agent is evaluated based on its fitness value. After computing the total force and acceleration, the velocity and position each agent are updated. These steps will be continued until stopping criteria is met and the best solution is returned by the algorithm.

Similar to other meta-heuristic algorithms, GSA also has some weaknesses such as having complex operators and taking long computational time.

Step 1: *Start.*
Step 2: *Randomized initialization.*
Step 3: *Fitness evaluation of agents.*
Step 4: *Update G(t), best(t), worst(t) and $M_i(t)$ for i = 1,2,...,N.*
Step 5: *Calculation of the total force in different directions.*
Step 6: *Calculation of acceleration and velocity.*
Step 7: *Updating agents' position.*
Step 8: *Repeat steps 3 to 7 until the stop criterion is reached.*
Step 9: *Return best solution.*

Fig. 10 GSA pseudo code

5.2 Population-based meta-heuristic algorithms in binary search space

The binary search space can be considered as a hypercube which a particle or an agent can move to nearer and farther its corners by flipping various numbers of the bits [79].

Each dimension has only a binary value of '0' or '1'. Therefore, the particle velocity can be considered by the number of bits changed per iteration, or the Hamming distance between the particle at time *t* and *t+l*. Consequently, updating the particle position happens when a switching between '0' and '1' values occurs.

Based on the mentioned rules, the binary versions of PSO and GSA have been proposed by Kennedy and Eberhart [79] and Rashedi et al. [80] respectively. In the next subsections, an overview of these algorithms is briefly represented to provide an appropriate background of binary search space.

5.2.1 PSO in binary search space (BPSO)

BPSO operates on discrete binary variables, therefore, the main difference between binary PSO and PSO is that the particle position has two values '0' or '1'. Hence, the velocity of particle in BPSO is calculated as PSO algorithm (Eq. (6)) and is transferred into a probability function in the interval of [0, 1] in order to update the particle position as shown in Eq. (38) and Eq. (39):

$$S\left(v_{id}\left(t+1\right)\right)=\frac{1}{1+e^{-v_{id}\left(t+1\right)}}, \tag{38}$$

$$\begin{aligned} &if \quad rand < S(v_{id}(t+1)) \quad then \quad x_{id}(t+1) = 1 \\ &else \quad x_{id}(t+1) = 0 \quad for \quad i = 1,2,\ldots,N \end{aligned}, \tag{39}$$

Where, $\vec{V_i} = (v_{i1}, v_{i2}, ..., v_{id})$ and $\vec{X_i} = (x_{i1}, x_{i2}, ..., x_{id})$ are the velocity and position of *ith* particle respectively, and *N* is the number of particles. For better convergence, $|v_{id}| < v_{max}$ is considered with $v_{max} = 6$.

Although BPSO has been applied in different discrete problems [81, 82, 83], its convergence rate is not good in many applications [84].

5.2.2 GSA in binary search space (BGSA)

In binary mode of GSA [80], updating the force, acceleration and velocity are similar to the GSA algorithm (Eq. (29) to Eq. (36)). Also, $R_{ij}(t)$ in the Eq. (30) is computed as hamming distance between two agents *i* and *j*, and *G* can be considered as a linear decreasing function:

$$G(t) = G_0\left(1 - \frac{t}{T}\right), \tag{40}$$

Where, G_0 is a constant value, *t* and *T* are current iteration and the total number of iteration respectively.

Moreover, the value of position is switched between '0' and '1' based on the velocity. Therefore, a probability function is defined to map the value of velocity to the range of [0, 1]. In other words, BGSA modifies the velocity according to Eq. (36) and the new position changes to either '0' or '1' based on the given probability of function as demonstrated in Eq.(41) and Eq. (42):

$$S(v_{id}(t+1)) = |\tanh(v_{id}(t+1))|, \tag{41}$$

$$\begin{aligned} &if \quad rand < S(v_{id}(t+1)) \quad then \quad x_{id}(t+1) = complement(x_{id}(t)) \\ &else \quad x_{id}(t+1) = x_{id}(t) \quad for \quad i = 1,2,\ldots,N \end{aligned}, \tag{42}$$

Similar to BPSO, in this context, $|v_{id}| < v_{max}$ and v_{max} =6 are considered.

Although the experiments of meta-heuristic algorithms have shown promising results and many researchers have tried to improve the performance of these algorithms in optimization problems [85, 86, 87, 88, 89], more research is still needed to overcome the disadvantages of these algorithms and to enhance their efficiency.

6 Discussion

Exact optimization algorithms are not efficient in solving large scale combinatorial and multimodal problems. In these problems, exhaustive search for the algorithms is impractical since the search space develops exponentially with the problem size. Hence, many researchers have applied meta-heuristic algorithms to solve the problems. These algorithms have many advantages to name a few:

1. They are robust and can adapt solutions with changing conditions and environment.
2. They can be applied in solving complex multimodal problems.
3. They may incorporate mechanisms to avoid getting trapped in local optima.
4. They are not problem-specific algorithm.
5. These algorithms are able to find promising regions in a reasonable time due to exploration and exploitation ability.
6. They can be easily employed in parallel processing.

Although the mentioned algorithms have obtained satisfactory results in various fields, they do not guarantee an optimal solution is ever found also they have some unavoidable disadvantages.

For instance, GA has the inherent drawbacks of prematurity convergence and unpredictable results. Also; it uses complex functions in selection and crossover operators and sometimes, the encoding scheme is difficult. PSO suffers from trapping into local optima and slow convergence speed, whereas GSA and ICA take long computational time to achieve the results. Furthermore, some of these algorithms have several parameters to tune and often parameters setting is a challenge for various optimization problems. Another noteworthy point is that many problems are expressed in a binary representation. In other words, some solutions are encoded binary form or some problems are binary in nature. Nevertheless, some meta-heuristic algorithms are designed for only continuous (real) or discrete (binary) search space and sometimes, they have good performance only for one of the search spaces. For example, ICA and the original of ACO have been designed for continuous and discrete search space respectively. Also, binary PSO has some inherent disadvantages such as poor convergence rate and failure to achieve desired results which bring about a decrease in performance of algorithm in the binary search space. Generally, the main disadvantages of using meta-heuristics are summarized as: trapping into local optima, slow convergence speed, long computational time, tuning many parameters, difficult encoding scheme and having good performance only in real or binary search spaces. Hence, the performance enhancement of previous meta-heuristics or even

introduction of new ones seems to be necessary. Table 2 provides the strengths and weaknesses of some well-known meta-heuristic algorithms.

Table 2: Advantages and disadvantages of some meta-heuristic algorithms

Algorithm	Advantages	Disadvantages
GA [49, 50 , 51]	The ability of: 1. Solving different kinds of optimization problems 2. Finding a global best solution in many problems 3. Easy to combining with other algorithms 4. Design for real and binary search space	1. Slow convergence rate 2. Finding sub-optimal solution 3. Unpredictable results 4. Dependency of the crossover and mutation rates on the stability and convergence. 5. Weak local search 6. Difficult encoding scheme
AIS [51]	1. It explores new search areas 2. Free of local optima 3. It is data driven self-adaptive methods.	1. Finding sub-optimal solution rather than the exact optimum solution 2. Too many parameter setting
SA [90, 91]	1. Low computational time 2. Free of local optima 3. Easy for implementation 4. Convergent property	Dependency of the solution quality on: 1. Maximum iteration number of the inner loop (cooling schedule) 2. Initial temperature
ACO [92, 93]	1. Scalability, robustness and flexibility in dynamic environments 2. Proper for graph-based problems	1. No easy to code. 2. Using trial and errors to parameters initializations 3. The original algorithm has been design for discrete search space 4. Difficult theoretical analysis
PSO [39, 54, 56, 94]	1. Simple structure 2. Easy to implement 3. Fast and cheap 4. Having few parameters to adjust 5. Efficient global search approach 6. Less dependent on initial points	1. Weak local search 2. Slow convergence rate and trapping into local optima when solving complex multimodal problems
ICA	1. Implementation of ICA code by author 2. Compatible in different problems	1. Using many equations and complex operators 2. Long computational time 3. Tuning many parameters 4. Design only for continuous (real) search space.
GSA	1. Implementation of GSA and BGSA code by author	1. Using complex operators 2. Long computational time

	2. Compatible in different problems

7 Conclusion

In this paper, we have presented and compared most important meta-heuristic algorithms. In the algorithms, several techniques have been considered to improve the performance of meta-heuristics. Meanwhile, none of meta-heuristic algorithms are able to present a higher performance than others in solving all problems. Also, existing algorithms suffer from some drawbacks such as slow convergences rate, trapping into local optima, having complex operators, long computational time, need to tune many parameters and design for only real or binary search space. Hence, proposing new meta-heuristic algorithms to minimizing the disadvantages is an open problem.

ACKNOWLEDGEMENTS

The authors would like to thank *Soft Computing Research Group (SCRG)*, Universiti Teknologi Malaysia (UTM), Johor Bahru Malaysia, for the support in making this study a success.

References

[1] Neapolitan, R., Naimipour K., *Foundations of Algorithms using C++ Pseudo code (3rd ed.)*, Jones and Bartlett, (2004).

[2] Jansson, C., Knoppel, O., "A branch and bound algorithm for bound constrained optimization problems without derivatives", *Journal of Global Optimization*, Vol. 7, (1995), pp. 297-331.

[3] Toroslu, I. H., Cosar, A. "Dynamic programming solution for multiple query optimization problem", *Information Processing Letters*, Vol. 92, (2004), pp. 149–155.

[4] Balev, S., Yanev, N., Fréville, A., Andonov, R., "A dynamic programming based reduction procedure for the multidimensional 0–1 knapsack problem", *European Journal of Operations Research*, Vol. 186, No. 1, (2008), pp. 63-76.

[5] Marti, R., Gallego, M., Duarte, A., "A branch and bound algorithm for the maximum diversity problem", *European Journal of Operations Research*, Vol. 200, (2010), pp. 36–44.

[6] Farmer, J. D., Packard, N. H., Perelson, A. S., "The immune system, adaptation and machine learning", *Physica D*, Vol. 2, (1986), pp. 187–204.

[7] Tang, K. S., Man, K. F., Kwong, S., He, Q., "Genetic algorithms and their applications", *IEEE Signal Processing Magazine,* Vol. 13, No. 6, (1996), pp. 22–37.

[8] Dorigo, M., Maniezzo, V., Colorni, A., "The ant system: optimization by a colony of cooperating agents", *IEEE Transactions on Systems, Man, and Cybernetics–Part B,* Vol. 26, No. 1, (1996), pp. 29–41.

[9] Kennedy, J., Eberhart, R., "Particle swarm optimization", *Proceedings of IEEE International Conference on Neural Networks,* (1995), pp. 1942–1948.

[10] Shi, Y., Eberhart, R., "A modified particle swarm optimizer", *Proceedings of IEEE International Conference on Evolutionary Computation*, (1998), pp. 69–73.

[11] Bishop, J. M., "Stochastic searching network", *Proceedings of 1st IEE Conference on Artificial Neural Networks*, (1989), pp. 329–331.

[12] Karaboga, D., "An idea based on honey bee swarm for numerical optimization", *Technical Report, TR06,* (2005).

[13] Shah-Hosseini, H., "Problem solving by intelligent water drops", *Proceedings of IEEE Congress on Evolutionary Computation,* (2007), pp. 3226–3231.

[14] Rabanal, P., Rodríguez, I., Rubio, F., "Using river formation dynamics to design heuristic algorithms", *In AKL et al. (Eds.) Unconventional Computation,* (2007), pp. 163–177, Springer-Verlag.

[15] Rashedi, E., Nezamabadi, S., Saryazdi, S., "GSA: a gravitational search algorithm", *Information Sciences,* Vol. 179, No. 13, (2009), pp. 2232– 2248.

[16] Kaveh, A., Talatahari, S., "A novel heuristic optimization method: charged system search", *ActaMechanica*, Vol. 213, (2010), pp. 267–289.

[17] Qasem, S. N., Shamsuddin, S. M., "Memetic Elitist Pareto Differential Evolution algorithm based Radial Basis Function Networks for classification problems", *Applied Soft Computing,* Vol. 11, No. 8, (2011), pp. 5565–5581.

[18] Qasem, S. N., Shamsuddin, S. M., "Radial basis function network based on time variant multi-objective particle swarm optimization for medical diseases diagnosis", *Applied Soft Computing,* Vol. 11, No. 1, (2011), pp. 1427–1438.

[19] Senaratne, R., Halgamuge, S., Hsu, A., "Face recognition by extending elastic bunch graph matching with particle swarm optimization", *Journal of Multimedia*, Vol. 4, No. 4, (2009), pp. 204–214.

[20] Cao, K., Yang, X., Chen, X., Zang, Y. Liang, J., Tian, J., "A novel ant colony optimization algorithm for large-distorted fingerprint matching", *Pattern Recognition,* Vol. 45, (2012), pp. 151–161.

[21] Beheshti, Z., Shamsuddin, S. M., Hasan, S., "MPSO: Median-oriented Particle swarm optimization", *Applied Mathematics and Computation*, Vol. 219, No. 11, (2013), pp. 5817–5836.

[22] Beheshti, Z., Shamsuddin, S. M., *Centripetal accelerated swarm particle optimization and its applications in machine learning*, PhD thesis, Universiti Teknologi Malaysia (UTM), (2013).

[23] Cordon, O., Damas, S., Santamaria, J., "A fast and accurate approach for 3D image registration using the scatter search evolutionary algorithm", *Pattern Recognition Letter*, Vol. 27, (2006), pp. 1191–1200.

[24] Lu, D. S., Chen, C. C., "Edge detection improvement by ant colony optimization", *Pattern Recognition Letter*, Vol. 29, (2008), pp. 416–425.

[25] Sousa, T., Silva, A., Neves, A., "Particle swarm based data mining algorithms for classification tasks", *Parallel Computing*, Vol. 30, (2004), pp. 767–783.

[26] Freitas, A. A., Timmis, J., "Revisiting the foundations of artificial immune systems for data mining", *IEEE Transactions on Evolutionary Computation*, Vol. 11, No. 4, (2007), pp. 521–537.

[28] Beheshti, Z., Shamsuddin, S. M., Yuhaniz, S. S., "Binary Accelerated Particle Swarm Algorithm (BAPSA) for discrete optimization problems", *Journal of Global Optimization*, (2012), In Press.

[29] Lazar, A., Reynolds, R. G., *Heuristic knowledge discovery for archaeological data using genetic algorithms and rough sets*, Artificial Intelligence Laboratory, Department of Computer Science, Wayne State University, (2003).

[30] Russell, S. J., Norvig, P., *Artificial Intelligence a Modern Approach*, New Jersey, Prentice Hall, (1995).

[31] Laporte, G., Osman, I. H., "Routing problems: A bibliography", *Annals Operations Research*, Vol. 61, (1995), pp. 227-262.

[32] Voss, S. Martello, S., Osman, I. H., Roucairol, C., *Meta-Heuristics - Advances and Trends in Local Search Paradigms for Optimization*, Kluwer Academic Publishers, (1999).

[33] Dreo, J., "Dreaming of Metaheuristics", *http://metah.nojhan.net*, (2007).

[34] Vaessens, R. J. M., Aarts, E. H. L., Lenstra, J. K., "A local search template", *Computers & Operations Research*, Vol. 25, No. 11, (1998), pp. 969-979.

[35] Birattari, M., Paquete, L., Stutzle, T., Varrentrapp, K., "Classification of Metaheuristics and Design of Experiments for the Analysis of Components", *Technical Report, AIDA-01-05*, (2001).

[36] Congram, R. K., Potts, C. N., Van de Velde, S. L., "An iterated dynasearch algorithm for the single-machine total weighted tardiness scheduling problem", *INFORMS Journal on Computing*, Vol. 14, No. 1, (2002), pp. 52-67.

[37] Glover, F., McMillan, C., "The general employee scheduling problem: an integration of MS and AI", *Computers & Operations Research*, Vol. 13, No. 5, (1986), pp. 563-573.

[38] Mladenović, N., Hansen, P., "Variable neighborhood search", *Computers & Operations Research*, Vol. 24, No. 11, (1997), pp. 1097–1100.

[39] Bonabeau, E. Dorigo, M., Theraulaz, G., *Swarm intelligence: from natural to artificial systems*, Oxford University Press, (1999).

[40] Tripathi, P. K., Bandyopadhyay, S., Pal, S. K., "Multi-Objective Particle Swarm Optimization with time variant inertia and acceleration coefficients", *Information Sciences*, Vol. 177, (2007), pp. 5033–5049.

[41] Voudouris, C., Tsang, E., "Partial constraint satisfaction problems and guided local search", *Proceedings of Second International Conference on Practical Application of Constraint Technology (PACT'96)*, (1996), pp. 337-356.

[42] Holland, J. H., *Adaptation in natural and artificial systems: an introductory analysis with applications to biology, control, and artificial intelligence*, Michigan, Ann Arbor, University of Michigan Press, 1975.

[43] Kirkpatrick, S., Gelatto, C. D., Vecchi, M. P., "Optimization by simulated annealing", *Science*, Vol. 220, (1983), pp. 671–680.

[44] Glover, F., "Tabu Search - Part 1", *ORSA Journal on Computing*, Vol. 1, No. 2, (1989), pp. 190–206.

[45] Glover, F., "Tabu Search - Part 2", *ORSA Journal on Computing*, Vol. 2, No. 1, (1990), pp. 4–32.

[46] Atashpaz-Gargari, E., Lucas, C., "Imperialist Competitive Algorithm: An algorithm for optimization inspired by imperialistic competition", *Proceedings of IEEE Congress on Evolutionary Computation*, (2007), pp. 4661–4667.

[47] Goldberg, D. E., Deb, K., "A comparative analysis of selection schemes used in genetic algorithms", *Proceedings of the First Workshop on Foundations of Genetic Algorithms*, (1990), pp. 69-93.

[48] Abraham, A., Nedjah, N., Mourelle, L. M., "Evolutionary Computation: from Genetic Algorithms to Genetic Programming", *Studies in Computational Intelligence (SCI)*, Vol. 13, (2006), pp. 1–20.

[49] Leung, Y., Gao, Y., Xu, Z. B., "Degree of population diversity - a perspective on premature convergence in genetic algorithms and its markov chain analysis", *IEEE Transaction on Neural Network*, Vol. 8, No. 5, (1997), pp. 1165-1176.

[50] Hrstka, O., Kučerová, A., "Improvements of real coded genetic algorithms based on differential operators preventing premature convergence", *Advances in Engineering Software*, Vol. 35, (2004), pp. 237–246.

[51] Moslemipour, G., Lee, T.S., Rilling, D., "A review of intelligent approaches for designing dynamic and robust layouts in flexible manufacturing systems", *International Journal of Advanced Manufacturing Technology*, Vol. 60, (2012), pp. 11–27.

[52] Shi, Y., Eberhart, R. C., "Empirical study of particle swarm optimization", *Proceedings of IEEE Congress on Evolutionary Computation*, (1999), pp. 1945–1950.

[53] Bratton, D. and Kennedy, J., "Defining a standard for particle swarm optimization", *Proceedings of the 2007 IEEE Swarm Intelligence Symposium*, (2007), pp. 120–127.

[54] Liang, J. J., Qin, A. K., Suganthan, P. N., Baskar, S., "Comprehensive learning particle swarm optimizer for global optimization of multimodal functions", *IEEE Transactions on Evolutionary Computation*, Vol. 10, No. 3, (2006), pp. 281–295.

[55] Zhan, Z.-H., Zhang, J., Li, Y., Chung, H.-S., "Adaptive Particle Swarm Optimization", *IEEE Transactions on Systems, Man, and Cybernetics–Part B*, Vol. 39, No. 6, (2009), pp. 1362-1381.

[56] Gao, W-F., Liu, S-Y., Huang, L-L., "Particle swarm optimization with chaotic opposition-based population initialization and stochastic search technique", *Communications in Nonlinear Science and Numerical Simulation*, Vol. 17, No. 11, (2012), pp. 4316–4327.

[57] Clerc, M., Kennedy, J., "The particle swarm-explosion, stability and convergence in a multidimensional complex space", *IEEE Transactions on Evolutionary Computation*, Vol. 6, No. 1, (2002), pp. 58–73.

[58] Trelea, I. C., "The particle swarm optimization algorithm: Convergence analysis and parameter selection", *Information Processing Letters*, Vol. 85, No. 6, (2003), pp. 317–325.

[59] Kadirkamanathan, V., Selvarajah, K., Fleming, P. J., "Stability analysis of the particle dynamics in particle swarm optimizer", *IEEE Transactions on Evolutionary Computation*, Vol. 10, No. 3, (2006), pp. 245–255.

[60] Yasuda, K., Ide, A., Iwasaki, N., "Stability analysis of particle swarm optimization", *Proceedings of the fifth Metaheuristics International Conference*, (2003), pp. 341–346.

[61] Bergh, F. V., Engelbrecht, A. P., "A study of particle optimization particle trajectories", *Information Sciences*, Vol. 176, No. 8, (2006), pp. 937–971.

[62] Eberhart, R. C., Shi, Y., "Particle swarm optimization: Developments, applications and resources", *Proceedings of IEEE Congress on Evolutionary Computation*, (2001), pp. 81–86.

[63] Li, X. D., Engelbrecht, A. P.: Particle swarm optimization, "An introduction and its recent developments", *Proceedings of IEEE conference companion on Genetic and evolutionary computation*, (2007), pp. 3391–3414.

[64] Kennedy, J., Mendes, R., "Population structure and particle swarm performance", *Proceedings of IEEE Congress on Evolutionary Computation*, (2002), pp. 1671–1676.

[65] Kennedy, J., Mendes, R., "Neighborhood topologies in fully informed and best-of-neighborhood particle swarms", *IEEE Transactions on Systems, Man, and Cybernetics Part-C,* Vol. 36, No. 4, (2006), pp. 515-519.

[66] Liang, J. J., Suganthan, P. N., "Dynamic multi-swarm particle swarm optimizer", *Proceedings of Swarm Intelligence Symposium*, (2005), pp. 124–129.

[67] Mendes, R., Kennedy, J., Neves, J., "The fully informed particle swarm: simpler, maybe better", *IEEE Transactions on Evolutionary Computation*, Vol. 8, No. 3, (2004), pp. 204–210.

[68] Eberhart, R. C., Shi, Y., "Fuzzy adaptive particle swarm optimization", *Proceedings of IEEE Congress on Evolutionary Computation*, (2001), pp. 101–106.

[69] Eberhart, R. C., Shi, Y., "Tracking and optimizing dynamic systems with particle swarms", *Proceedings of IEEE Congress on Evolutionary Computation*, (2001), pp. 94–97.

[70] Eberhart, R. C., Shi, Y. H., "Comparing inertia weights and constriction factors in particle swarm optimization", *Proceedings of IEEE Congress on Evolutionary Computation*, (2000), pp. 84–88.

[71] Ratnaweera, A., Halgamuge, S., Watson, H., "Self-organizing hierarchical particle swarm optimizer with time-varying acceleration coefficients", *IEEE Transactions on Evolutionary Computation*, Vol. 8, No. 3, (2004), pp. 240–255.

[72] Angeline, P. J., "Using selection to improve particle swarm optimization", *Proceedings of the 1998 IEEE International Conference on Evolutionary Computation*, (1998), pp. 84–89.

[73] Juang, C. F., "A hybrid of genetic algorithm and particle swarm optimization for recurrent network design", *IEEE Transactions on Systems, Man, and Cybernetics Part-C*, Vol. 34, No. 2, (2004), pp. 997–1006.

[74] Zhan, Z.-H., Zhang, J., Li, Y., Shi, Y.-H., "Orthogonal Learning Particle Swarm Optimization", *IEEE Transactions on Evolutionary Computation,* Vol. 15, No. 6, (2011), pp. 832-847.

[75] Biabangard-Oskouyi, A., Atashpaz-Gargari, E., Soltani, N., Lucas, C., "Application of imperialist competitive algorithm for materials property characterization from sharp indentation test", *International Journal of Engineering Simulation*, Vol. 1, No. 3, (2009), pp. 337-355.

[76] Rajabioun, R., Hashemzadeh, F., Atashpaz-Gargari, E., Mesgari, B., Salmasi, F.R., "Identification of a MIMO evaporator and its decentralized PID controller tuning using Colonial Competitive Algorithm", *Proceedings of the 17th World Congress, the International Federation of Automatic Control*, (2008), pp. 9952-9957.

[77] Atashpaz-Gargari, E., Hashemzadeh, F., Rajabioun, R., Lucas, C., "Colonial Competitive Algorithm, a novel approach for PID controller design in MIMO distillation column process", *International Journal of Intelligent Computing and Cybernetics*, Vol. 1, No. 3, (2008), pp. 337–355.

[78] Schutz, B., *Gravity from the ground up*, Cambridge University Press, (2003).

[79] Kennedy, J., Eberhart, R. C., "A discrete binary version of the particle swarm algorithm", *Proceedings of IEEE international conference on computational cybernetics and simulation*, (1997), pp. 4104–4108.

[80] Rashedi, E., Nezamabadi, S., Saryazdi, S., "BGSA: binary gravitational search algorithm", *Natural Computing*, Vol. 9, No. 3, (2010), pp. 727–745.

[81] Kong, M., Tian, P., "Apply the particle swarm optimization to the multidimensional knapsack problem", *In Rutkowski, L., Tadeusiewicz, R., Zadeh, L. A., Zurada, J. M. (Eds.) Artificial Intelligence and Computational Intelligence*, (2006), pp. 1140–1149, Springer-Verlag.

[82] Chuang, L.-Y., Chang, H.-W., Tu, C.-J., Yang, C.-H., "Improved binary PSO for feature selection using gene expression data", *Computational Biology and Chemistry*, Vol. 32, No. 1, (2008), pp. 29-38.

[83] Mezmaz, M., Melab, N., Kessaci, Y., Lee, Y.-C., Talbi, E.-G. Zomaya, A.Y., Tuyttens, D., "A parallel bi-objective hybrid metaheuristic for energy-aware scheduling for cloud computing systems", *Journal of Parallel and Distributed Computing*, Vol. 71, No. 11, (2011), pp. 1497–1508.

[84] Nezamabadi-pour, H., Rostami Shahrbabaki, M., Maghfoori-Farsangi, M., "Binary Particle Swarm Optimization: Challenges and new Solutions", *CSI Journal on Computer Science and Engineering, in Persian*, Vol. 6, No. 1, (2008), pp. 21-32.

[85] Upadhyaya, S. and Setiya, R., "Ant colony optimization: a modified version", *International Journal of Advances* in *Soft Computing and Its Applications*, Vol. 1, No. 2, (2009), pp. 77-90.

[86] Chen, W.-N., Zhang, J., Chung, H. S. H., Zhong, W.-L., Wu, W.-G., Shi, U.-H., "A novel set-based particle swarm optimization method for discrete optimization problems", *IEEE Transactions on Evolutionary Computation*, Vol. 14, No. 2, (2010), pp. 278-300.

[87] Bagher R. M. and Payman, J., "Water delivery optimization program, of jiroft dam irrigation networks by using genetic algorithm", *International Journal of Advances* in *Soft Computing and Its Applications*, Vol. 1, No. 2, (2009), pp. 151-155.

[87] Juang, Y. T., Tung, S.-L., Chiu, H.-C., "Adaptive fuzzy particle swarm optimization for global optimization of multimodal functions", *Information Sciences*, Vol. 181, (2011), pp. 4539–4549.

[88] Premalatha, K., Natarajan, A. M., "Hybrid PSO and GA Models for Document Clustering", *International Journal of Advances* in *Soft Computing and Its Applications*, Vol. 2, No. 3, (2010), pp. 302-320.

[89] Hemanth, D. J., Vijila, C. K. S. and Anitha, J., "Performance Improved PSO based Modified Counter Propagation Neural Network for Abnormal MR Brain Image Classification", *International Journal of Advances* in *Soft Computing and Its Applications*", Vol. 2, No. 1, (2010), pp. 65-84.

[90] Romeo, F., Sangiovanni-Vincentelli, A., "A theoretical framework for simulated annealing", *Algorithmica*, Vol. 6, (1991), pp. 302–345.

[91] Johnson, D. S., Aragon, C. R., McGeoch, L.A., Schevon, C., "Optimization by simulated annealing—an experimental evaluation; part 2, graph-coloring and number partitioning", *Operations Research*, Vol. 39, No. 3, (1991), pp. 378-406.

[92] Dorigo, M., Birattari, M., Stützle, T., "Ant colony optimization – artificial ants as a computational intelligence technique", *IEEE Computational Intelligence Magazine*, (2006), pp. 28-39.

[93] Dorigo, M., Socha, K, "An Introduction to Ant Colony Optimization", *In Gonzalez, T., F. (Ed.) Approximation Algorithms and Metaheuristics*, (2007), pp. 1-19, CRC Press.

[94] Chan, F. T. S., Tiwari, M. K., *Swarm intelligence: focus on ant and particle swarm optimization*, InTech, (2007).