Contents lists available at ScienceDirect

Journal of Computational Science

journal homepage: www.elsevier.com/locate/jocs

Rain-fall optimization algorithm: A population based algorithm for solving constrained optimization problems

S. Hr. Aghay Kaboli[a,*], J. Selvaraj[a], N.A. Rahim[a,b,**]

[a] University Malaya Power Energy Dedicated Advanced Centre (UMPEDAC), Level 4, Wisma R&D UM, JalanPantai Baharu, 59990 Kuala Lumpur, Malaysia
[b] Renewable Energy Research Group, King Abdulaziz University, Jeddah 21589, Saudi Arabia

ARTICLE INFO

Article history:
Received 18 May 2016
Received in revised form
15 December 2016
Accepted 29 December 2016
Available online 3 January 2017

Keywords:
Natural inspired computation
Iterated-based algorithms
Rain-fall
Metaheuristic approaches
Economic dispatch

ABSTRACT

This paper proposes rain-fall optimization algorithm (RFO), a new nature-inspired algorithm based on behavior of raindrops, for solving of real-valued numerical optimization problems. RFO has been developed from a motivation to find a simpler and more effective search algorithm to optimize multi-dimensional numerical test functions. It is effective in searching and finding an optimum solution from a large search domain within an acceptable CPU time. Statistical analysis compared the solution quality with well-known heuristic search methods. In addition, an economic dispatch (ED) optimization problem is run on an IEEE 30-bus test system, and the results, compared with those of recent optimization methods, show RFO performing relatively well, sufficiently effective to solve engineering problems. The constraint-handling strategy of the proposed method for solving ED problem is to generate and work with feasible solutions along all the optimization iterations without any mismatch between electricity demand and the total amount of power generation. Unlike the penalty methods, this strategy is unaffected by parameter setting of applied optimization method and its applicability for solving constrained optimization problems is not hampered. Eventually, its robustness is validated by the results of a sensitivity analysis of the parameters.

1. Introduction

Metaheuristic optimization methods are widely applied in searching for, and finding, approximate optimum solutions in large-scale optimization problems. The term “metaheuristic” refers to general-purpose algorithms applicable to optimization problems, usually without much modification to adapt to a given/specific problem. Most metaheuristic methods implement some form of stochastic optimization so their solutions depend on the set of random variables generated. A metaheuristic method will be successful on a given optimization problem if it can provide a balance between the exploitation of the accumulated search experience and the exploration of the search space to identify regions with high quality solutions in a problem specific, near optimal way. The main difference between the existing metaheuristic methods concerns the particular way in which they try to achieve this balance [1]. A graphical representation of how metaheuristics could be classified is show in Fig. 1. The metaheuristic approaches can be characterized by different aspects in to the different classes such as:

i Trajectory vs. discontinuous methods
ii Population-based vs. single-point search
iii Memory usage vs. memoryless methods
iv Single vs. multiple neighborhoods structures
v Dynamic vs. static objective function
vi Nature-inspired vs. non-nature inspiration

The computational drawbacks such as too many control parameters, excessive sensitivity to the initial value of these parameters, premature convergence, and time-consuming computation that are frequently encountered in different classes of metaheuristic optimization methods have encouraged researchers to develop naturally inspired metaheuristic optimization methods to solve large-scale optimization problems. Naturally inspired metaheuristic optimization methods are generic population-based optimization techniques that converge toward an optimum solution through stochastic exploration. In general, naturally inspired

* Corresponding author.
** Corresponding author at: University Malaya Power Energy Dedicated Advanced Centre (UMPEDAC), Level 4, Wisma R&D UM, JalanPantai Baharu, Kuala Lumpur 59990, Malaysia.
E-mail addresses: kaboli@siswa.um.edu.my (S.Hr. Aghay Kaboli), nasrudin@um.edu.my (N.A. Rahim).

http://dx.doi.org/10.1016/j.jocs.2016.12.010
1877-7503/© 2016 Elsevier B.V. All rights reserved.

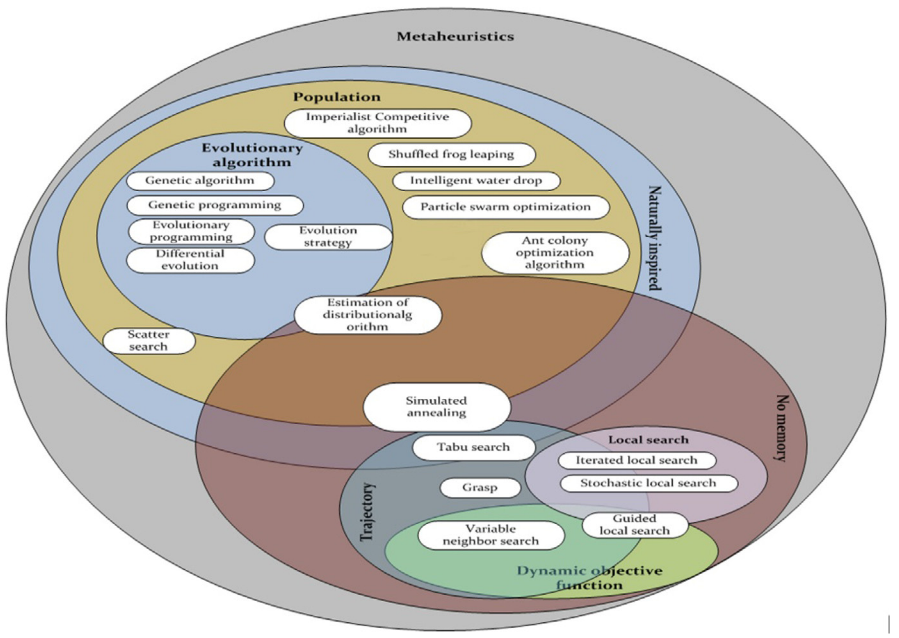

Fig. 1. Different classifications of metaheuristics approaches.

optimization algorithms are clustered into evolutionary algorithms (EAs), particle swarm inspired algorithms, socio-human or socio-political inspired algorithms, and natural-phenomena-inspired algorithms.

EAs mimic the evolution of biological population, with an ability to adapt to environmental changes when finding optimum value by natural selection. Genetic algorithm (GA), the first evolutionary-based optimization method, was developed by John Holland [2]. Its capability to reach near-optimum value has increased its application in large problems. Despite it, GAs have a long processing time in evolving a near-optimum solution to the global optimum [3]. Furthermore, not all problems will tend to optimum solution through GAs [4]; other EAs are therefore proposed: differential evolutionary (DE) [5], evolutionary strategy (ES) [6,7] and backtracking search algorithm (BSA) [8,9].

Particle swarm inspired algorithms, known as swarm intelligence (SI), were inspired by animal herding, bird flocking, ant colonies, fish schooling and bacterial growth, i.e., the collective behavior of self-organized and decentralized systems. Typically, particle swarm inspired algorithms mimic a flock of animals that have no leader find food randomly and following the group member closest to the food source (potential solution). The methods include particle swarm optimization (PSO) [10,11], ant colony optimization (ACO) [12], artificial bee colony algorithm (ABC) [13], group search optimizer (GSO) [14], artificial cooperative search algorithm (ACS) [15], bat algorithm (BA) [16] and shuffled frog-leaping algorithm (SFLA) [17].

Optimization algorithms inspired by humans' creative problem-solving processes should be superior to the optimization algorithms inspired by the collective behavior of other creatures [18]. Owing to this superiority, socio-human or socio-political inspired algorithms have attracted more attention for their application in large-scale problems. Such optimization methods include brain storm optimization (BSO) [18], anarchic society optimization (ASO) [19], imperialist competitive algorithm (ICA) [20], league championship algorithm (LCA) [21], social emotional optimization algorithm (SEOA) [22] and society civilization algorithm [23]. A common feature among them is the moving of a population of possible solutions into promising areas of the search space, in terms of the problem's objective, while seeking the optimum without any leadership, through social behavior. This is in contrast with particle-swarm-inspired algorithms and EAs, which each converges toward the optimum solution according to social intelligence and evolution.

Most of natural phenomena occur optimally. Hence natural-phenomena-inspired algorithms such as artificial physics optimization (APO) [24], intelligent water drop (IWD) [25], simulated annealing [26], black hole (BH) [27], electro-magnetism optimization (EMO) [28], charged system search (CSS) [29], gravitational search algorithm (GSA) [30], harmony search (HS) [31], spiral optimization [32], optimization of Brownian motion of gases (GBMO) [33], and atmosphere clouds model optimization (ACMO) [34] mimic natural phenomena in finding better solutions:. Most of natural-phenomena-inspired algorithms are based on physical, chemical, and biological concepts. These optimization methods converge to the optimum solution neither by evolutionary progress nor by SI.

The objective of this paper is to present a new naturally inspired optimization algorithm based on raindrops trickling according to physical gravitational law. The proposed algorithm is modeled on raindrop flow over a mountainous surface. The movement of the raindrops' population is based on the surface slope, hence simple movement of the population members is proposed as effective exploration of the search space of the optimization problem.

The rest of the paper is organized as follows: First, the basic aspect, flowchart, and characteristics of RFO algorithm are described. Next, application of RFO to numerical test functions, discussion of results and performance analysis are presented. In Section 4, the performance of RFO is evaluated on a real world optimization problem and the obtained results are compared with other well-known optimization methods. Eventually, Section 5 outlines the conclusion.

2. Proposed method description

RFO mimics the natural behavior of raindrops trickling down a hill (high position) towards a valley (low position). In this section, the method's concept and calculation processes are described.

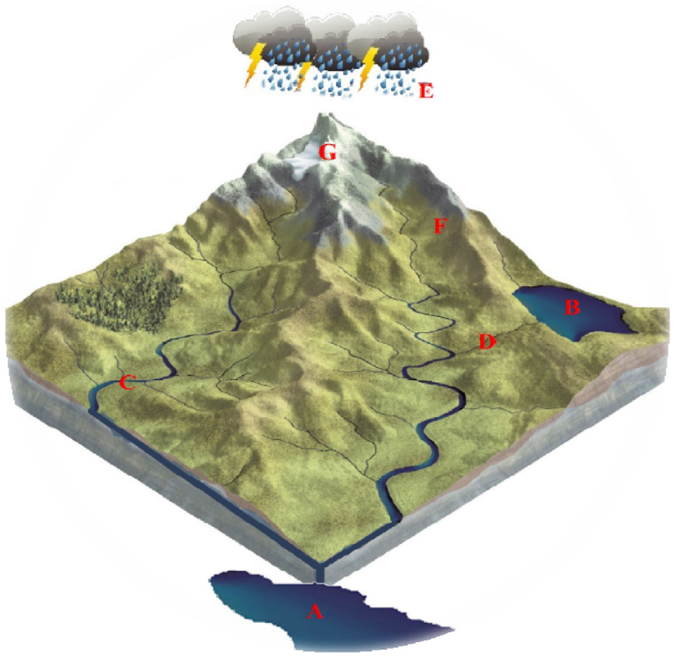

Fig. 2. Rain-fall schematic.
A: Sea (global optimum), B: Lake (local optimum), C: River, D: Tributary, E: Rain-fall, F: Valley, G: Melting snow.

2.1. Basic of rain-fall algorithm

Most of natural-inspired algorithms imitate the natural biological and social behavior of creatures. RFO drew inspiration from raindrop behavior. Raindrops naturally trickle down along a slope from a peak then form the rivers and always reach to the lowest land points or empty out into the sea. As shown in Fig. 2, most rivers are fed by rainwater running off the land (rain-fall), or by melting snow and ice in the summit of mountains. The rivers begin life as a tiny stream running down a mountain slope. The raindrops follow cracks and folds in the land as they flow downhill. Small streams formed by raindrops meet and join together, growing larger and larger until the flow can be called a river. As the water flows downhill, it might being trapped in the puddles and shapes a lake (local optimal). On its way down, the water shapes the landscape by wearing away rock and carving out a network of valleys. Reaching lower ground, the river widens and takes a winding route. Eventually, most rivers reach to the sea level (global optimal) and empty out into the sea. The way the proposed method converges from a guess to the optimum is similar with raindrop trickling from a peak to sea level in a mountainous area according to gravitation. As always raindrop tend to select the path having the steeper slope, RFO simulates this tendency and utilizes the gradient of the objective function to determine the solution that is superior to a guess.

To find the deepest valley and then reach to the sea level (global optimal), a population of raindrops is generated randomly at first iteration. The positions of the neighbor-points of every drop (the points around it) are compared with the drop's position before the drop moves towards the neighbor point occupying the lowest position. This movement continues until the drop reaches the valley. As long as all the drops are trickling down from an upper to a lower position, even if there are puddles on the drops' way to the valley, they can still overflow and emerge from the puddles to keep moving towards the valley by an appropriate mechanism implemented by the RFO algorithm.

The movement of the particle in the proposed method is similar to gradient based optimization methods and that of traditional single-point algorithms such as hill-climbing (HC) and gradient-descent. Hill-climbing adjusts only a single element of the optimizing variables in each attempt to determine whether the change improves the value of the objective function. Gradient-descent optimizes by adjusting all variables according to the gradient of the objective function. Unlike in the former, in the latter the objective function needs to be continuous and differentiable but instead greatly narrows the range of real-world applications. RFO is a population-based optimization algorithm comprising a set of points (population) that solves a problem through a merit order list among all the population members. It reaches near optimum neither by limiting change to only one element at each attempt nor by depending on the mathematical form of the objective functions, since it moves from a guess to the superior solution by random search around the guess.

Three possible alternative versions of HC; steepest-ascent hill-climbing (SAHC), next-ascent hill-climbing (NAHC), and random-mutation hill-climbing (RMHC), have been proposed in [35,36]. However, HCs are still known as greedy local search methods. Different local search algorithms such as random walks and simulated annealing proposed to overcome the drawback associated with HCs, but the applicability of simulated annealing is problem-specific because it relies on finding lucky jumps that improve the position. In problems that involve more dimensions, the cost of finding such a jump may increase exponentially with dimensionality.

However, the applicability of RFO is not problem-specific since it basically is a population-based optimization algorithm and utilizes the reduction of the objective function value to determine a superior solution from a solution for the next iteration. The proposed optimization technique begins with a randomly generated solutions and then making successive random search around current point for modifications the current value until a stopping criterion. The procedure of RFO is independent of the first guess which allows it to converge to the near optimal via any form of guessing. In other words, it can start with any initial solution and improves on it to find optimal solution with a high probability.

2.2. Rain-fall algorithm

Fig. 3 is the flowchart of the proposed algorithm. As with other evolutionary methods, RFO starts with an initial population (raindrops). These are the relevant terms that should be explained before describing the proposed algorithm as follows:

Raindrop: a particle in a population; it is a vector containing the variables of the optimization problem and satisfying the problem constraints; if the population size is m, the drop number i is defined as in Eq. (1).

$$\mathbf{D}^i = [x_{i,1}\, x_{i,2}\, x_{i,3} \ldots x_{i,k} \ldots x_{i,n}] \quad i \in \{1, 2, 3, \ldots, m\} \tag{1}$$

Where:
n: number of optimization variables,
$x_{i,k}$: kth variable of optimization problem,
$\mathbf{D}^i$: drop number i

Note that $\mathbf{D}^i$ is assumed a point in N-dimensional axes, i.e., it can be referred to as an N-element vector, or a point in the N-dimensional axes. Rain-fall deals with raindrops during an optimization process. It is generated according to uniform random distribution function and subject to all the constraints in Eq. (2).

$$x_{i,k} = U(low_k,\ up_k) \tag{2}$$

Where:
low_k and up_k: lower and upper limits of x_k,
U: uniform distribution function

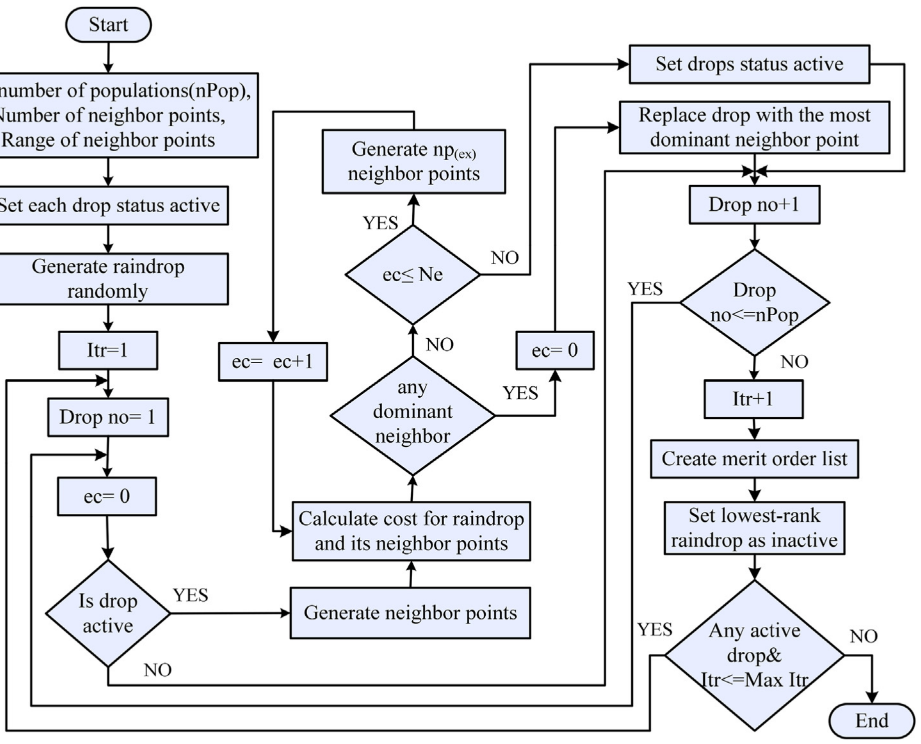
Start
Set number of populations(nPop), Number of neighbor points, Range of neighbor points
Set each drop status active
Generate raindrop randomly
Itr=1
Drop no= 1
ec= 0
Is drop active
YES
NO
Generate neighbor points
Calculate cost for raindrop and its neighbor points
any dominant neighbor
NO
YES
ec≤ Ne
YES
NO
Generate np(ex) neighbor points
ec= ec+1
ec= 0
Set drops status active
Replace drop with the most dominant neighbor point
Drop no+1
Drop no<=nPop
YES
NO
Itr+1
Create merit order list
Set lowest-rank raindrop as inactive
Any active drop& Itr<=Max Itr
YES
NO
End

Fig. 3. Flowchart of the proposed algorithm (Rain-fall).

Neighborhood: since the raindrop **D** includes N elements and represents a point in N-dimensional axes, the domain with radius vector of **r** around the point is called the neighborhood; upon change in the raindrop value, the neighborhood is updated.

Neighbor point: a point in drop's neighborhood, generated randomly during optimization; the neighbor point j of drop i is represented by $\mathbf{NP}_j^i$ according to the following formula:

$$\begin{aligned}&\|(\mathbf{D}^i-\mathbf{NP}_j^i).\bar{\mathbf{u}}_k\| \le \|\mathbf{r}.\bar{\mathbf{u}}_k\|\\ &i=\{1,2,3,\ldots,m\}\\ &j=\{1,2,3,\ldots,np\}\\ &k=\{1,2,3,\ldots,n\}\\ &r=r_{\text{initial}}\times f(\text{iteration})\end{aligned} \tag{3}$$

Where:

r: the real positive vector representing the neighborhood size in an N-dimensional space, the higher its elements are, the bigger the neighborhood size is.

$\mathbf{r}_{initial}$: the initial neighborhood size and f(iteration) is a function to adapt the step size [37], it adapts the neighborhood size within iterations.

np: the number of neighbor points to be generated during calculation.

$\hat{\mathbf{U}}_k$: the unit vector of kth dimension.

Dominant drop: among all the neighbor points of drop $\mathbf{D}^i$, the dominant neighbor point $\mathbf{NP}_d^i$ is a point that satisfies (4). The objective function is represented by **F** so, the values of the functions for drop and its neighbor points respectively can be obtained through $\mathbf{F}(\mathbf{D}^i)$ and $\mathbf{F}(\mathbf{NP}_j^i)$.

$$\begin{aligned}&\mathbf{F}(\mathbf{NP}_d^i) < \mathbf{F}(\mathbf{D}^i)\\ &\mathbf{F}(\mathbf{NP}_d^i) < \mathbf{F}(\mathbf{NP}_j^i)\; j\in\{1,2,3,\ldots,np\}-\{d\}\end{aligned} \tag{4}$$

Active drop: a drop that has a dominant neighbor point.

Inactive drop: a drop that has no dominant neighbor point.

Explosion process: may occur during optimization when there is no dominant neighbor point for a raindrop such that the raindrop's status is inactive; this is due to either insufficient number of neighbor points or to the drop being stuck in a search domain rather than reaching a local/global minimum; the explosion process is carried out so the drop gets out from this situation; the process is assumed to be applied Ne times; the number of neighbor points to be generated in the explosion process ($\text{np}_{(\text{ex})}$) is determined by Eq. (5).

$$\text{np}_{(\text{ex})} = \text{np}\times\text{eb}\times\text{ec} \tag{5}$$

Where np is the number of neighbor points in normal condition (with no explosion), eb is the explosion base (indicating the explosion range), and ec is the explosion counter.

Raindrops rank: in each iteration of the optimization process, the ranks of all the raindrops are calculated according to Eq. (6), for use in the merit order list.

$$C1_t^i = \mathbf{F}(\mathbf{D}^i)|_{at\ t^{th}\ iteration} - \mathbf{F}(\mathbf{D}^i)|_{at\ 1^{st}\ iteration}$$

$$C2_t^i = \mathbf{F}(\mathbf{D}^i)|_{at\ t^{th}\ iteration}$$

$$Rank_t^i = \omega_1\times order(C1_t^i)+\omega_2\times order(C2_t^i) \tag{6}$$

where:

$C1_t^i$: the absolute change in objective function from the first iteration at iteration t for raindrop $\mathbf{D}^i$.

$C2_t^i$: the value of objective function for raindrop $\mathbf{D}^i$ at iteration t.

Order($C1_t^i$)and Order($C2_t^i$): orders of C1 and C2 at iteration t when they are sorted in ascending order,

ω_1 and ω_2: weighting coefficients which are assumed to be 0.5,

$Rank_t^i$: the rank of raindrop $\mathbf{D}^i$ at iteration t.

Merit order list: a list containing the ranks of raindrops in each iteration, in ascending order; each drop is removed from the population or given special rights (such as higher number of explosions associated with the drop according to its position in the merit order list) for consideration in optimization.

In the first iteration, raindrops are generated randomly and fall over cost function as natural raindrops on a mountain. Then each drop allocates a neighborhood for itself and in that neighborhood specific numbers of neighbor points are generated randomly considering the problem constraints. The proposed method works with feasible solutions along all the optimization iterations and only feasible solutions are generated within the search mechanism of this algorithm. If a neighbor point during the optimization process, overflows the allowed search-space limits as a result of RFO's search strategy, then the neighbor point generated beyond the search-space limits is updated using the following equation:

$$\text{if } (\mathbf{NP}_j^{\,i})_k < \text{low}_k \text{ then } (\mathbf{NP}_j^{\,i})_k = \text{low}_k$$

$$\text{or} \tag{7}$$

$$\text{if } (\mathbf{NP}_j^{\,i})_k > \text{up}_k \text{ then } (\mathbf{NP}_j^{\,i})_k = \text{up}_k$$

Next, the cost function is calculated for the drop and its neighbor points. The value obtained for the raindrop is then compared with the value of its neighbor points to find the dominant neighbor point (the one with the lowest cost function). If there is any dominant neighbor point, then the raindrop moves toward that neighbor point and changes its position. Otherwise, the raindrop is at the minimum point or is stuck (through insufficient number of neighbor points). If the latter, new and sufficient numbers of neighbor points are generated to overcome the problem (this process in the algorithm is called "explosion").

The first iteration of the RFO algorithm is completed when the same procedure applies to all the raindrops. It continues until either the iteration reaches the maximum number or all the drops reach the minimum points.

The main parameters of this method are the number of neighbor points, the neighborhood size, the number of population, and the number of neighbor points in the explosion process. The values of the parameters depend on the problem size and type, e.g., the more the optimization variables, the bigger the population should be. The RFO algorithm is as follows:

1- Start.

2- Set the following parameters:

1. nPop: population size
2. np: number of neighbor points of each drop
3. maximum number of iterations

3- Generate the first population of raindrops including nPop (populations size) rain drops randomly so each raindrop satisfies the constraints.

4- Set to 1, the Iteration value.

5- By default, set to Active, all the drops' status.

6- Do the following on each active rain drop:

1. Generate np neighbor points
2. Obtain the cost function values of drops and their neighbor points.
3. If there is a dominant neighbor point then change the drop's present position to that point otherwise, apply explosion process to the drop.
4. If there is no dominant neighbor point after Ne times of explosion, set the drop's status to Inactive.

7- Create a merit-order list and remove specific numbers of low-ranking drops or assign higher Ne to high-ranking drops.

8- Set Iteration = Iteration + 1;

9- If there is any Active raindrop and the iterations have not reached the maximum allowed number, go to step 6.

10- Calculate the cost function values of all the raindrops.

11- Find the raindrop with the minimum cost function.

12- Print the raindrop position and cost as the optimum solution.

13- End.

3. Application of RFO to numerical optimization problems

This section includes application of proposed method to test functions, discussion of results and performance analysis.

3.1. Benchmark functions

To evaluate the performance of any optimization algorithm, it should be compared with other algorithms and with well-known benchmark functions applied. In this study, ten common benchmark functions as tabulated in Table 1 are considered to assess the performance of proposed method.

Some of the functions evaluating the ability of the proposed RFO algorithm are called unimodal (e.g., Sphere and Rosenbrock), each with only one minimum. The other functions with many local minima are categorized in a multimodal group, where trapping the algorithm in local minima is easy because the number of local minima increases exponentially with increase in the function dimension. Ackley and Griewank are two well-known multimodal functions whose local minima are regularly distrusted. In some multimodal functions, an optimization method might not achieve the optimum value, owing to the proximity of the local minima to the global minima or the small amount of the global minima of the benchmark relative to the search space, e.g., the Kowalik function. The rest of the benchmarks, i.e., Six-Hump Camel-Back, Goldstein-Price Hartman, and Shekel functions, are multimodal test problems with a low number of dimensions and a few local minima.

3.2. Experimental result and discussion

To evaluate the performance of the proposed method, the results of three evolutionary algorithms (namely, GA, PSO, and GSO) were compared with the results of [14]. The parameters setting of proposed optimization method have been set as follows:

The initial neighborhood size considered according to Eq. (8).

$$r_k = [\text{up}_k - \text{low}_k] \times 2\% \tag{8}$$

Where:

k represents the kth variable of optimization problem.

The number of neighbor points considered according to Eq. (9).

$$\text{np} = 20 \times \text{n} \tag{9}$$

Where:

n represents the total number of optimization variables.

The initial number of neighbor points to be generated in the explosion process considered according to Eq. (10).

$$\text{np}_{(\text{ex})} = \text{np} \times 10 \tag{10}$$

The code was written in MATLAB (R2010b) and executed on a personal computer, with a core-2 quad processor of 2.6 GHz clock speed and 2 GB RAM. The algorithm ran more than 1000 trials for the first four benchmarks known as high-dimension benchmark functions; for the rest, over 50 trials. The average function value, standard deviation of each benchmark function, and calculation time are listed in Table 2, which shows that on the high-dimension benchmark functions GA performed the worst, though it required the shortest calculation time among all the algorithms tested. Comparison between PSO and rain-fall showed that RFO algorithm

Table 1
Ten Benchmark Functions.

Name of the function	Test function	n	Search space	Global minimum
Sphere	$F_1(x) = \sum_{i=1}^{n} x_i^2$	30	$[-100, 100]^n$	0
Rosenbrock	$F_2(x) = \sum_{i=1}^{n-1} (100(x_{i+1} - x_i^2)^2 + (x_i - 1)^2$	30	$[-30, 30]^n$	0
Ackley	$F_3(x) = -20\exp\left(-0.2\sqrt{\frac{1}{n}\sum_{i=1}^{n} x_i^2}\right) - \exp\left(\frac{1}{n}\sum_{i=1}^{n}\cos 2\pi x_i\right) + 20 + e$	30	$[-30, 30]^n$	0
Griewank	$F_4(x) = \frac{1}{4000}\sum_{i=1}^{n} x_i^2 - \prod_{i=1}^{n}\cos\left(\frac{x_i}{\sqrt{i}}\right) + 1$	30	$[-600, 600]^n$	0
Kowalik	$F_5(x) = \sum_{i=1}^{11}\left[a_i - \frac{x_1(b_i^2 + b_i x_2)}{b_i^2 + b_i x_3 + x_4}\right]^2$	4	$[-5, 5]^n$	0.0003075
Six-Hump Camel-Back	$F_6(x) = 4x_1^2 - 2.1_1^4 + \frac{1}{3}x_1^6 + x_1x_2 - 4x_2^2 + 4x_2^4$	2	$[-5, 5]^n$	−1.0316285
Goldstein-Price	$F_7(x) = [1 + (x_1 + x_2 + 1)^2(19 - 14x_1 + 3x_1^2 - 14x_2 + 6x_1x_2 + 3x_2^2)] \times [30 + (2x_1 + 1 - 3x_2)^2(18 - 32x_1 + 12x_1^2 + 48x_{2-} - 36x_1x_2 + 27x_2^2]$	2	$[-2, 2]^n$	3
Hartman	$F_8(x) = -\sum_{i=1}^{4} c_i \exp\left[-\sum_{j=1}^{4} a_{ij}(x_j - p_{ij})^2\right]$	4	$[0, 1]^n$	−3.86
Hartman	$F_9(x) = -\sum_{i=1}^{4} c_i \exp\left[-\sum_{j=1}^{6} a_{ij}(x_j - p_{ij})^2\right]$	6	$[0, 1]^n$	−3.32
Shekel	$F_{10}(x) = -\sum_{i=1}^{10} [(x - a_i)(x - a_i)^T + c_i]^{-1}$	4	$[0, 10]^n$	−10

generating remarkably better solution than did PSO except on the Sphere benchmark function. On benchmark F_2, rain-fall algorithm performed better than did GSO, though GSO produced better results on three other high-dimension benchmarks. In terms of calculation time, rain-fall algorithm was faster than GSO and PSO except on the third test function (Ackley). On low-dimension functions, rain-fall algorithm consistently outperformed the other algorithms. Furthermore, the lowest standard deviation was observed on rain-fall algorithm for all the benchmarks except Hartman Family (F_8) and Sheckel's Family (F_{10}).

To further assess performance, its results were compared with those of [14,38], which included fast evolutionary programming (FEP), classical evolutionary programming (CEP), fast evolution strategy (FES), and conventional evolution strategy (CES) methods. The mean function value and standard deviation are as in Table 3, which shows, in terms of solution accuracy and algorithm reliability, that rain-fall algorithm is superior to three other algorithms on most of the test functions. In Rosenbrock and Griwank test functions, FEP performed better than did the others.

3.3. *Performance analysis*

Rain-fall algorithm is an iterated-based algorithm that generates raindrops randomly. Neighbor points are then generated to produce movement in each raindrop. The number of neighbor points, step size, and number of explosions are the determining parameters in each iteration, so their effect on solution quality, robustness, computation time, and convergence toward optimum value were investigated. The performance of the proposed method was evaluated through the experiment results presented in this section.

3.3.1. *The importance of neighbor point numbers*

Table 4 illustrates the effects of neighbor points on the optimum solution for 50% and 100% increment over the benchmark functions F_1–F_{10} listed in Table 1. The results show that for high dimensional functions, as the number of neighbor points increases, the quality of the solution increases; CPU time, however, increases too, owing to increased computational burden.

This statement is still valid for low dimensional test function, Shckel's Family (F_{10}), for 50% increase in the number of neighbor points. Unlike the aforementioned benchmark functions, the results on Goldstein-Price and Hartman (F_8) were better: lower CPU time. The worst results for CPU time were obtained on Kowalik, Camel-back and Hartman (F_9) test function. Incrementing the number of neighbor points for high-dimension benchmark functions increased the probability of finding better results from among the randomly generated neighbor points. On low-dimension functions, incrementing the number of neighbor point did not influence the results.

3.3.2. *The effectiveness of number of explosion*

Table 5 illustrates the effect of 50% and 100% increased number of explosions. With increase in number of explosions, the quality of the solution improves and the execution time increases, on both the high-dimension and the low-dimension test functions. The reason is that more explosions force the drops to move toward the global minima during the search process.

3.3.3. *The effectiveness of step size*

Table 6 illustrates the effect of 50% and 100% increased step-size. The larger the step size, the more the solution quality is sacrificed to speed. Among all the test functions, Sphere relatively showed more sensitivity to step-size variation. Finding an exact global optimum on a multimodal function seems difficult without prior

Table 2
Average fitness value of benchmark functions.

Function	Algorithm	Mean	Std.	CPU Times (s)
Sphere	GA	3.1711	1.6621	12.5
	PSO	3.6927e-37	2.4598e-36	36.6
	GSO	1.9481e-8	1.1629e-8	27.6
	Rain-fall	**1.4607e-6**	1.9387e-7	21.33
Rosenbrock	GA	338.5616	361.497	11.3
	PSO	37.3582	32.1436	37.6
	GSO	49.8359	30.1771	27.8
	Rain-fall	**32.4053**	32.9	11.67
Ackley	GA	0.8678	0.2805	19.3
	PSO	1.3404e-3	4.2388e-2	44.1
	GSO	2.6548e-5	3.0820e-5	34.4
	Rain-fall	**1.724e-4**	**1.5976e-5**	**51.53**
Griewank	GA	1.0038	6.7545e-2	16.1
	PSO	0.2323	0.4434	45.4
	GSO	3.0792e-2	3.0867e-2	35.9
	Rain-fall	**9.86e-2**	**7.81e-2**	**28.9**
Kowalik	GA	7.0878e-3	7.8549e-3	15.3
	PSO	3.8074e-4	2.5094e-4	63.5
	GSO	3.7713e-4	2.5973e-4	36.6
	Rain-fall	**3.1553e-4**	**2.8845e-5**	**21.19**
Camel-Back	GA	−1.0298	3.1314e-3	0.5
	PSO	−1.0160	1.2786e-2	0.3
	GSO	−1.031628	0	0.2
	Rain-fall	**−1.031628**	0	**0.09**
Goldstein-Price	GA	7.5027	10.3978	1
	PSO	3.0050	1.2117e-3	2.3
	GSO	3.0	0	1.5
	Rain-fall	**3.0**	**0**	**0.83**
Hartman Family	GA	−3.8624	6.2841e-4	0.3
	PSO	−3.8582	3.2127e-3	0.9
	GSO	−3.8628	3.8430e-6	0.5
	Rain-fall	**−3.8628**	**3.3785e-5**	**0.39**
Hartman Family	GA	−3.2627	6.0398e-2	0.7
	PSO	−3.1846	6.1053e-2	1.8
	GSO	−3.2697	5.9647e-2	1.0
	Rain-fall	**−3.3051**	**7.1e-3**	**1.3**
Shckel's Family	GA	−4.9108	3.4871	1.4
	PSO	−8.9439	1.6304	2.8
	GSO	−7.4022	3.2131	1.9
	Rain-fall	**−10.1134**	**1.6880**	**2.81**

Table 3
Mean function value and standard deviation of FEP, CEP, FES, CES and proposed method (Rain-fall).

Function	FEP	CEP	FES	CES	Rain-fall
Sphere	5.7e-4 (1.3e-4)	2.2e-4 (5.9e-4)	2.5e-4 (6.8e-4)	3.4e-5 (8.6e-6)	**1.4607e-6** **(1.9387e-7)**
Rosenbrock	5.06 (5.87)	6.17 (13.61)	33.28 (43.13)	6.69 (14.45)	**32.40** **(32.9)**
Ackley	1.8e-2 (2.1e-2)	9.2 (2.8)	1.2e-2 (1.8e-3)	9.07 (2.84)	**15e-4** **(1.5976e-5)**
Griewank	1.6e-2 (2.2e-2)	8.6e-2 (0.12)	3.7e-2 (5.0e-2)	0.38 (0.77)	**9.86e-2** **(7.81e-2)**
Kowalik	5.0e-4 (3.2e-4)	4.7e-4 (3.0e-4)	9.7e-4 (4.2e-4)	1.2e-3 (1.6e-5)	**3.1664e-4** **(2.8845e-5)**
Camel-back	−1.03 (4.9e-4)	−1.03 (4.9e-4)	−1.0316 (6.0e-7)	−1.0316 (6.0e-7)	**−1.031628** **0**
Goldstein-Price	3.02 (0.11)	3.0 (0)	3.0 (0)	3.0 (0)	**3.000** **(0)**
Hartman	−3.86 (1.4e-5)	−3.86 (1.4e-2)	−3.86 (4.0e-3)	−3.86 (1.4e-5)	**−3.8628** **(3.3785e-5)**
Hartman	−3.27 (5.9e-2)	−3.28 (5.8e-2)	−3.23 (0.12)	−3.24 (5.7e-2)	**−3.3051** **(7.1e-3)**
Shckel's Family	−6.57 (3.14)	−9.1 (2.92)	−7.63 (3.27)	−8.5 (1.25)	**−10.1134** **(1.6880)**

Table 4
The mean function value, standard deviation and CPU time of different benchmarks with 50% and 100% increment neighbor point number.

Function	50%			100%		
	Mean	Std. Dev	CPU Times (s)	Mean	Std. Dev	CPU Times (s)
Sphere	1.3843e − 6	1.6666e − 6	23.60	1.3461e − 6	1.7866e − 7	28
Rosenbrock	32.24	32.8397	12.4	31.98	31.6428	13.4
Ackley	1.5993e − 4	1.4729e − 5	63.4	1.5694e − 4	1.3086e − 5	74.5
Griewank	9.78e − 2	7.28e − 2	30.2	8.74e − 2	5.84e − 2	34.3
Kowalik	3.4339e − 4	1.0662e − 4	16.7	4.204e − 4	1.6874e − 4	11.4
Camel-back	−1.031627	0	0.09	−1.031625	0	0.08
Goldstein-Price	3.00	5.8299e − 5	0.22	3.00	1.3210e − 4	0.20
Hartman	−3.8627	1.4243e − 4	0.32	−3.8626	9.809e − 5	0.26
Hartman	−3.2940	1.9e − 2	1.3	−3.2750	1.8e − 2	1.26
Shckel's Family	−10.5324	1e − 2	3.2	−10.4186	0.8	3.6

Table 5
The mean function value, standard deviation and CPU time of different benchmarks with50% and 100% increment explosion number.

Function	50%			100%		
	Mean	Std. Dev	CPU Times (s)	Mean	Std. Dev	CPU Times (s)
Sphere	1.3941e − 6	1.7888e − 7	23.5	1.3339e − 6	1.6575e − 7	27.2
Rosenbrock	31.97	28.4714	12.20	31.89	25.0489	13.1
Ackley	1.6097e − 4	1.2105e − 5	58.5	1.5953e − 4	1.4191e − 5	64.2
Griewank	8.66e − 2	5.90e − 2	32	8.63e − 2	5.72e − 2	35
Kowalik	3.1537e − 4	2.93e − 5	24.7	3.1483e − 4	2.8352e − 5	27.4
Camel-back	−1.031628	0	0.11	−1.031628	0	0.13
Goldstein-Price	3.00	0	1.11	3.00	0	1.4
Hartman	−3.8628	1.4283e − 5	0.5	−3.8628	1.4581e − 5	0.5
Hartman	−3.3062	5.3e − 3	2.1	−3.3074	8.9e − 3	2.6
Shckel's Family	−10.30	1.1102	3.4	−10.5349	4e − 2	3.54

Table 6
The mean Function value, standard deviation and CPU time of different benchmarks with50% and 100% increment of step size.

Function	50%			100%		
	Mean	Std. Dev	CPU Times (s)	Mean	Std. Dev	CPU Times (s)
Sphere	0.4163	4.4e − 2	8.8	0.7427	7.0e − 2	6.7
Rosenbrock	39.11	28.3055	8.2	48.05	27.2639	6.40
Ackley	2.5376e − 4	2.3880e − 5	49.5	3.4371e − 4	2.5033e − 5	43.70
Griewank	0.522	8.24e − 2	14.6	0.702	7.59e − 2	10.8
Kowalik	3.1752e − 4	3.2818e − 5	27.3	3.1246e − 4	2.3386e − 5	32.3
Camel-back	−1.031628	0	0.12	−1.031628	0	0.15
Goldstein-Price	3.00	0	1.3	3.00	0	1.5
Hartman	−3.8628	1.7359e − 5	0.4	−3.8628	8.8723e − 6	0.5
Hartman	−3.3077	5.4e − 3	2	−3.3070	5.5e − 3	2.4
Shckel's Family	−10.24	1.3656	1.35	−9.8558	1.9	0.84

knowledge of the function. Finding the global optimum by increasing the neighborhood size could take longer in limited search space (e.g., F_5–F_9).

To validate the robustness and reliability of the proposed algorithm, convergence graphs were plotted for all the test benchmarks, against variations in number of neighbor points, number of explosions, and step size. Fig. 4 illustrates the robustness of the proposed method, confirmed by either guessing or varying the parameters. Powerful searching ability within the neighborhood domain prevents premature convergence by parameter variation on all the benchmarks. The effect of step-size on the optimum solution was insignificant on all the benchmark functions. A possible conclusion is that as step-size increases, convergence improves and execution time shortens; contrarily for test functions with limited search domain, execution time lengthens.

4. Application of RFO in real-world optimization problem

In this section, the proposed method is applied for solving economic dispatch (ED) problem in power system as a real word optimization problem and its results are evaluated.

4.1. Economic dispatch (ED)

Economic dispatch (ED) is among the important issues considered an optimization problem in power system operation [39]. Its objective is to most economically divide the power demand among available units to reduce the total generation cost within operational constraints [40]. ED's objective function is the cost summation of N generating units. In order to consider the practical economic dispatch, transmission loss and non-convex cost function for each generating unit are considered. A sinusoidal term is added to the quadratic function in order to consider the non-convexity of generator cost function as Eq. (11). Fig. 5 illustrates non-convex generation cost function according to Eq. (11) for one and two generators.

$$\mathrm{F}_i(P_i) = a_i P_i^2 + b_i P_i + c_i + |\, e_i \times \sin\left(f_i\left(P_i^{\min} - P_i\right)\right)| \tag{11}$$

Where:
P_i: real output power of the i^{th} generator,
$F_i(P_i)$: the generation cost of the i^{th} generator,
$\mathrm{a_i}$, $\mathrm{b_i}$, $\mathrm{c_i}$, $\mathrm{e_i}$, and $\mathrm{f_i}$: cost coefficients of the $\mathrm{i^{th}}$ generator.

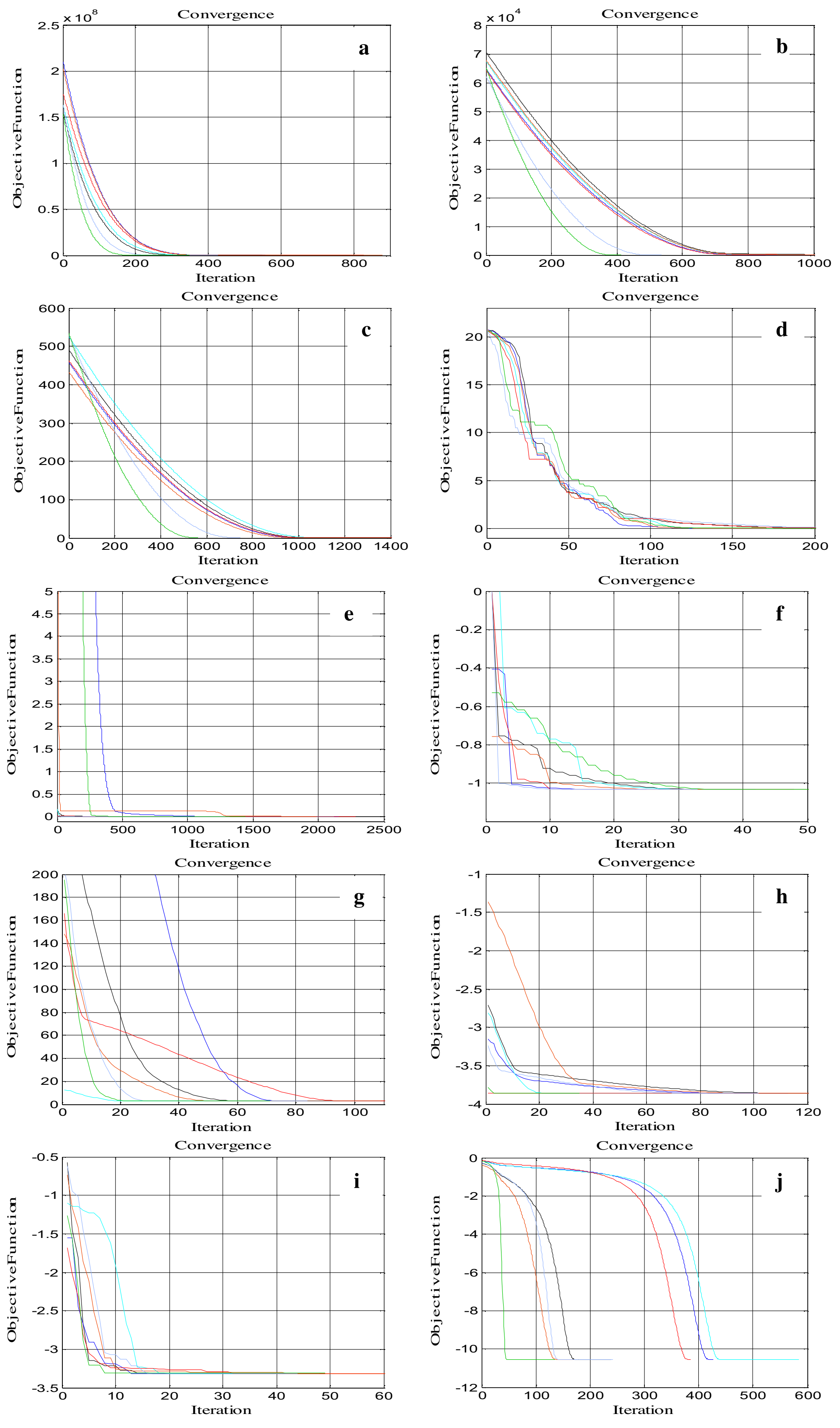

Fig. 4. Sensitivity analyses and convergence graph for 50% and 100% increment of np, Ne and r for 10 benchmark functions. (a): Sphere; (b): Rosenbrock; (c): Ackley; (d): Griewank; (e): Kowalik; (f): Camel-back; (g): Goldstein-Price; (h): Hartman; (i): Hartman; (j): Shckel's Family. Black: base case; Blue: 50% increment of np; Cyan: 100% increment of np; Red: 50% increment of Ne; Brown: 100% increment of Ne; Light blue: 50% increment of r; Green: 100% increment of r. (For interpretation of the references to colour in this figure legend, the reader is referred to the web version of this article.)

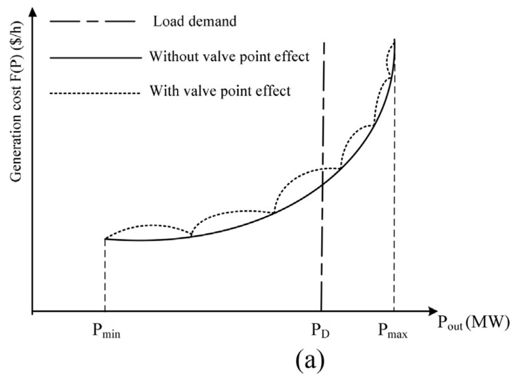

Fig. 5. Non-convex generation cost function for one power unit (a) and two power units (b).

The objective function in ED problem is minimizing the summation of all the generators' costs, as in Eq. (12).

$$MinF_c = \sum_{i=1}^{N} F_i(P_i) = F(\mathbf{P}) \tag{12}$$
$$\mathbf{P} = [P_1, P_2, ..., P_N]$$

The constraints of the economic dispatch problem comprise the following:

i. Power balance constraint:

The total power generation should meet the demand(P_D)and the real transmission loss(P_L).

$$\sum_{i=1}^{N} P_i - P_D - P_L = 0 \tag{13}$$

The transmission loss (P_L)formula with B matrix coefficients can be formulated as Eq. (14).

$$P_L = \sum_{i=1}^{N}\sum_{j=1}^{N} P_i B_{ij} P_J + \sum_{i=1}^{N} B_{0i} P_i + B_{00} \tag{14}$$

Where B_{00}, B_{0i}, and B_{ij} are transmission loss coefficients.

ii. Generator output limit constraint:

The real output power of each generator should be within lower and upper limits.

$$P_i^{\min} \le P_i \le P_i^{\max} \tag{15}$$

Where $P_i^{\min}$and $P_i^{\max}$are respectively the lower and upper limits of the ith generator.

To solve ED optimization problem with loss considered an N-element vector containing the output power of N generating units satisfying the constraints are generated. This vector in kth iteration of optimization process is:

$$\mathbf{P}(k) = [P_1^k, P_2^k, ..., P_N^k] \tag{16}$$

To generate this vector, its first (N-1) elements are generated randomly according to Eq. (15) and the last element is calculated in such a way that Eq. (13) is satisfied. This equation makes the last element (P_N) a dependent variable. The value of P_N is calculated by Eq. (17), which is derived by substituting Eq. (14) into Eq. (13).

$$C_2 P_N^2 + C_1 P_N + C_0 = 0$$

$$C_2 = B_{NN}$$

$$C_1 = 2 \times \sum_{i=1}^{N-1} P_i B_{iN} + B_{0N} - 1 \tag{17}$$

$$C_0 = \sum_{i=1}^{N-1}\sum_{j=1}^{N-1} P_i B_{ij} P_j + \sum_{i=1}^{N-1} (B_{0i} - 1) P_i + B_{00} + P_D$$

Solving the polynomial Eq. (17) produces P_N as the last element of the generation vector according to Eq. (18) (note that the positive root should be selected). If the value of this element is within the boundary specified in constraint (14) then the generation vector is valid, otherwise the procedure should be repeated.

$$P_N = \frac{-C_1 \pm \sqrt{C_1^2 - 4C_2 C_0}}{2C_2}, \quad C_1^2 - 4C_2 C_0 \ge 0 \tag{18}$$

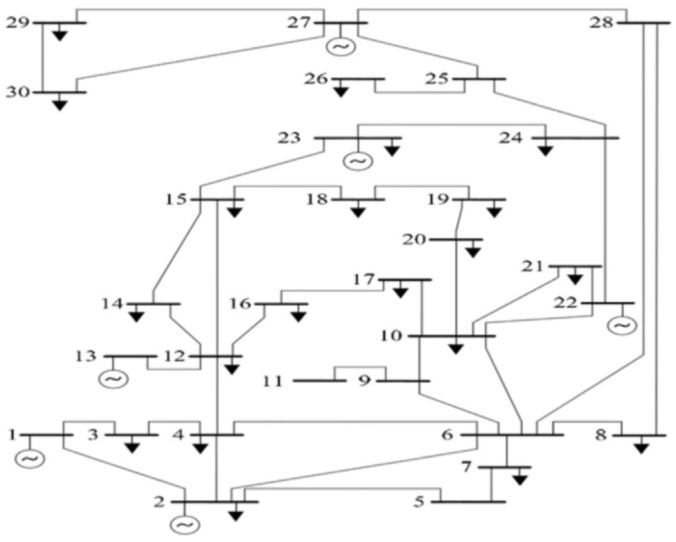

Fig. 6. Modified IEEE 30 bus test system with six generators.

4.2. Experimental result and discussion

In addition, to validate the proposed method's solving of ED optimization problems, it was applied to a modified IEEE 30-bus test system with six power generators (units) as shown in Fig. 6. The generation cost function for this power system is non-convex and the total power demand is 283.4 MW, while the transmission network loss are considered. The parameters for the ED problem are as

Table 7
The optimum solution of loss −considered economic dispatch for IEEE 30-bus test system.

Method	(MW)								Total generation cost ($/h)	CPU Times (s)
	P_1	P_2	P_3	P_4	P_5	P_6	Total Power	Loss		
GA	150.724	60.870	30.896	14.214	19.489	15.915	292.110	8.706	996.037	0.578
GA-APO	133.981	37.216	37.768	28.350	18.792	38.052	294.160	10.756	1101.491	0.156
NSOA	182.478	48.352	19.855	17.137	13.668	12.349	293.839	10.439	984.936	0.150
PSO	197.865	50.337	15.000	10.000	10.000	12.000	295.202	11.802	925.758	0.353
MSG-HP	199.633	20.000	23.762	18.393	17.102	15.692	294.583	11.183	925.640	0.621
Rain-fall	199.600	20.000	23.856	18.725	18.604	13.703	294.488	11.088	925.638	0.265

Table 8
The minimum, mean, and maximum total generation costs for loss–considered economic dispatch in the IEEE 30-bus test system.

Method	Max generation cost ($/h)	Mean generation cost ($/h)	Min generation cost ($/h)
GA	1117.13	–	996.04
GA-APO	1101.49	–	996.04
NSOA	992.48	–	984.94
PSO	928.43	926.39	925.76
MSG-HP	928.6	926.85	925.64
Rain-fall	926.01	925.76	925.64

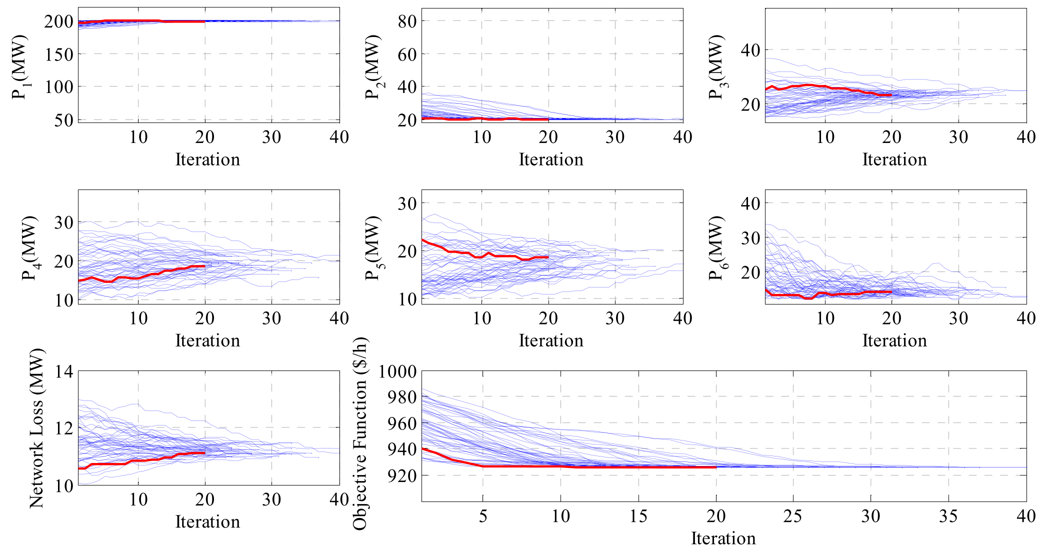

Fig. 7. Convergence of P_1-P_6, objective function, and transmission loss to optimal values within 50 trial (blue: 50 trial, red: best solution in 50 trial. (For interpretation of the references to colour in this figure legend, the reader is referred to the web version of this article.)

in [41]. For this case study, the proposed method was run 50 trials to enable statistical analysis of the results and for fair comparison of its computation efficiency, solution quality, and convergence characteristics. The simulation results and the statistical comparison with GA, GA-APO, NSOA [41], PSO and MSG-HP [42] in solving the ED problem are as presented in Table 7 and Table 8, respectively. The results show the proposed method to be capable of producing a higher-quality solution than most of the evolutionary methods with zero power mismatch. Fig. 7 shows converging output powers of the generating units and the objective function in 50 trials, as analyzed of the proposed method's robustness and convergence. Although the guess was generated randomly in each trial, the proposed algorithm was still able to converge it to optimal values. During optimization, the values of P_1–P_5 were generated randomly and the value of P_6 was calculated according to Eq. (18). Calculation of the transmission loss was based on the value of the problem variables according to Eq. (14). Fig. 7 indicates the convergence of the objective function to optimum value. The proposed method converged to the optimum value in about 25 iterations within 0.35 s. However, the minimum number of iterations for convergence to optimal values through 50 trials was 14 iterations within 0.14 s.

5. Conclusion

In this paper, rain-fall optimization algorithm (RFO) has been introduced as a new naturally inspired algorithm based on behavior of raindrops, for solving constrained optimization problems with an effective approach for seeking an optimum solution in a search domain. The process considers the numerical differential of the cost function rather than the mathematical computation of the gradient. RFO can be used on functions that are not continuous or differentiable, so it is considered as random-search, derivative-free, or black-box optimization method. An effective dynamic neighbor search method is also developed to facilitate the optimization process; it prevents premature convergence and being trapped in the local optima. As it works with feasible solutions along all the optimization iterations, only feasible solutions are generated within the local search mechanism of this algorithm. To evaluate the performance of the proposed method, the results of different well-known metaheuristic optimization algorithms were compared. The results, compared with those of recent optimization methods, indicate that RFO is effective, fast, and capable of solving various engineering optimization problems.

Acknowledgement

This work has been supported by HIR (high impact research) secretariat at university of Malaya through the "Campus network smart grid system for energy security" project (under grant number: H-16001-00-D000032).

References

[1] M. Birattari, L., Paquete, T. Strutzle, and K. Varrentrapp, Classification of Metaheuristics and Design of Experiments for the Analysis of Components Tech. Rep. AIDA-01-05, 2001.
[2] J.H. Holland, Adaptation in Natural and Artificial Systems: An Introductory Analysis with Applications to Biology, Control and Artificial Intelligence, U Michigan Press, 1975.
[3] E. Elbeltagi, T. Hegazy, D. Grierson, Comparison among five evolutionary-based optimization algorithms, Adv. Eng. Inform. 19 (2005) 43–53.
[4] A. Rafieerad, A. Bushroa, B. Nasiri-Tabrizi, A. Fallahpour, J. Vadivelu, S. Musa, et al., GEP-based method to formulate adhesion strength and hardness of Nb PVD coated on Ti–6Al–7Nb aimed at developing mixed oxide nanotubular arrays, J. Mech. Behav. Biomed. Mater. 61 (2016) 182–196.
[5] R. Sarker, S. Elsayed, and T. Ray,; Differential Evolution with Dynamic Parameters Selection for Optimization Problems, 2012.
[6] P. Gorman, M. Gregory, and D. Werner, Design of Ultra-Wideband, Aperiodic Antenna Arrays With the CMA Evolutionary Strategy, 2013.
[7] H.-G. Beyer, H.-P. Schwefel, Evolution strategies–a comprehensive introduction, Nat. Comput. 1 (2002) 3–52.
[8] M. Modiri-Delshad, S.H. Aghay Kaboli, E. Taslimi-Renani, N.A. Rahim, Backtracking search algorithm for solving economic dispatch problems with valve-point effects and multiple fuel options, Energy 116 (Part 1) (2016) 637–649, 12/1/2016.
[9] M. Modiri-Delshad, N.A. Rahim, Multi-objective backtracking search algorithm for economic emission dispatch problem, Appl. Soft Comput. 40 (2016) 479–494.
[10] T.T. Ngo, A. Sadollah, J.H. Kim, A cooperative particle swarm optimizer with stochastic movements for computationally expensive numerical optimization problems, J. Comput. Sci. 13 (March 31) (2016) 68–82.
[11] A.R. Rafieerad, A.R. Bushroa, B. Nasiri-Tabrizi, S.H.A. Kaboli, S. Khanahmadi, Ahmad Amiri, J. Vadivelu, F. Yusof, W.J. Basirun, K. Wasa, Toward improved mechanical, tribological, corrosion and in-vitro bioactivity properties of mixed oxide nanotubes on Ti–6Al–7Nb implant using multi-objective PSO, J. Mech. Behav. Biomed. Mater. 69 (May) (2017) 1–18.
[12] T. Liao, K., Socha, M. Montes de Oca, T. Stuetzle, and M. Dorigo, Ant Colony Optimization for Mixed-Variable Optimization Problems.
[13] M. Singh, B.K. Panigrahi, A.R. Abhyankar, Optimal coordination of electro–mechanical–based overcurrent relays using artificial bee colony algorithm, Int. J. Bio-Insp. Comput. 5 (2013) 267–280.
[14] S. He, Q.H. Wu, J.R. Saunders, Group search optimizer: an optimization algorithm inspired by animal searching behavior, Evol. Comput. IEEE Trans. on 13 (2009) 973–990.
[15] S.H.A. Kaboli, J. Selvaraj, N.A. Rahim, Long-term electric energy consumption forecasting via artificial cooperative search algorithm, Energy 115 (Part 1) (2016) 857–871, 11/15/2016.
[16] A.H. Gandomi, X.-S. Yang, Chaotic bat algorithm, J. Comput. Sci. 5 (2014) 224–232, 3//2014.
[17] M. Eusuff, K. Lansey, F. Pasha, Shuffled frog-leaping algorithm: a memetic meta-heuristic for discrete optimization, Eng. Optim. 38 (2006) 129–154.
[18] Y. Shi, An optimization algorithm based on brainstorming process, Int. J. Swarm Intell. Res. 2 (2011) 35–62.
[19] A. Ahmadi-Javid, Anarchic society optimization: a human-inspired method, in evolutionary computation (CEC), 2011 EEE Congress on (2011) 2586–2592.
[20] E. Atashpaz-Gargari, C. Lucas, Imperialist competitive algorithm: an algorithm for optimization inspired by imperialistic competition, Evolutionary Computation,. CEC 2007. IEEE Congress On, 2007 (2007) 4661–4667.
[21] A. Husseinzadeh Kashan, An efficient algorithm for constrained global optimization and application to mechanical engineering design: league championship algorithm (LCA), Comput.-Aided Des. 43 (2011) 1769–1792.
[22] Y. Xu, Z. Cui, J. Zeng, Social emotional optimization algorithm for nonlinear constrained optimization problems, in: Swarm, Evolutionary, and Memetic Computing, Springer, 2010, pp. 583–590.
[23] T. Ray, K.M. Liew, Society and civilization: an optimization algorithm based on the simulation of social behavior, Evol. Comput. IEEE Trans. 7 (2003) 386–396.
[24] L. Xie, Y. Tan, J. Zeng, Z. Cui, Artificial physics optimisation: a brief survey, Int. J. Bio-Insp. Comput. 2 (2010) 291–302.
[25] H. Shah-Hosseini, The intelligent water drops algorithm: a nature-inspired swarm-based optimization algorithm, Int. J. Bio-Insp. Comput. 1 (2009) 71–79.
[26] S. Kirkpatrick, Optimization by simulated annealing: quantitative studies, J. Stat. Phys. 34 (1984) 975–986.
[27] A. Hatamlou, Black hole: a new heuristic optimization approach for data clustering, Inform. Sci. 222 (February 10) (2013) 175–184.
[28] E. Cuevas, D. Oliva, D. Zaldivar, M. Pérez-Cisneros, H. Sossa, Circle detection using electro-magnetism optimization, Inform. Sci. 182 (2012) 40–55.
[29] A. Kaveh, S. Talatahari, A novel heuristic optimization method: charged system search, Acta Mech. 213 (2010) 267–289.
[30] E. Rashedi, H. Nezamabadi-Pour, S. Saryazdi, GSA: a gravitational search algorithm, Inform. Sci. 179 (2009) 2232–2248.
[31] Z.W. Geem, J.H. Kim, G. Loganathan, A new heuristic optimization algorithm: harmony search, Simulation 76 (2001) 60–68.
[32] K. Tamura, K. Yasuda, Spiral dynamics inspired optimization, JACIII 15 (2011) 1116–1122.
[33] M. Abdechiri, M.R. Meybodi, H. Bahrami, Gases brownian motion optimization: an algorithm for optimization (GBMO), Appl. Soft Comput. 13 (2013) 2932–2946.
[34] G.-W. Yan, Z.-J. Hao, A novel optimization algorithm based on atmosphere clouds model, Int. J. Comput. Intell. Appl. 12 (2013).
[35] S. Forrest, M. Mitchell, Relative building-block fitness and the building-block hypothesis, Ann. Arbor 1001 (1993) 48109.
[36] D.M.G. Taborda, L. Zdravkovic, Application of a Hill-Climbing technique to the formulation of a new cyclic nonlinear elastic constitutive model, Comput. Geotechnol. 43 (2012) 80–91, 6//2012.
[37] M. Schumer, K. Steiglitz, Adaptive step size random search, IEEE Trans. Autom. Control 13 (1968) 270–276.
[38] Y. Xin, L. Yong, L. Guangming, Evolutionary programming made faster, Evolu. Comput. IEEE Trans. on 3 (1999) 82–102.
[39] M. Modiri-Delshad, S.H.A. Kaboli, E. Taslimi, J. Selvaraj, N. Rahim, An iterated-based optimization method for economic dispatch in power system, Clean Energy and Technology (CEAT), 2013 IEEE Conference on (2013) 88–92.
[40] M. Modiri-Delshad, S. Koohi-Kamali, E. Taslimi, S.H.A. Kaboli, N. Rahim, Economic dispatch in a microgrid through an iterated-based algorithm, Clean Energy and Technology (CEAT) 2013 IEEE Conference on (2013) 82–87.
[41] T. Nadeem Malik, A. ul Asar, M.F. Wyne, S. Akhtar, A new hybrid approach for the solution of nonconvex economic dispatch problem with valve-point effects, Electr. Power Syst. Res. 80 (2017) 1128–1136, 9//2010.
[42] C. Yaşar, S. Özyön, A new hybrid approach for nonconvex economic dispatch problem with valve-point effect, Energy 36 (2011) 5838–5845, 10//2011.

S. Hr. Aghay Kaboli received the B.Eng degree in electrical engineering from IAU, Esfahan, Iran, in 2009 and the M.Eng. degree from University Malaya (UM), Kuala Lumpur, Malaysia, in 2012. Since 2011, he has been a High Impact Research Assistant for Campus Network Smart Grid for Energy Securit project with University Malaya, in the UM Power Energy Dedicated Advanced Center (UMPEDAC). He is a reviewer for IEEE Transactions, Elsevier and Springer journals. He also has been serving as an editor for American Journal of Electrical Power & Energy Systems, International Journal of Computer Systems (IJCS), and a guest editor for the special issue "Some Novel Algorithms for Global Optimization & Relevant Subjects" under the journal "Applied and Computational Mathematics (ACM)". His research interests include Artificial Intelligence (AI), smart grid, and power electronic.

J. Selvaraj received the B.Eng. (Hons.) degree from Multimedia University, Cyberjaya, Malaysia, in 2002, the M.Sc. degree in power electronics and drives jointly from the University of Birmingham, Birmingham, U.K., and the University of Nottingham, Nottingham, U.K., in 2004, and the Ph.D. degree from the University of Malaya, Kuala Lumpur, Malaysia, in 2009. He is currently an Associate Professor with UM Power Energy Dedicated Advanced Center (UMPEDAC). His research interests are single- and three-phase multilevel inverters, digital PI current-control techniques, photovoltaic inverters, and dc–dc converters.

N.A. Rahim received the B.Sc. (Hons.) and M.Sc. degrees from the University of Strathclyde, Glasgow, U.K., and the Ph.D. degree from Heriot–Watt University, Edinburgh, U.K., in 1995. He is currently a Professor with the University of Malaya, Kuala Lumpur, Malaysia, where he is also the Director of the UM Power Energy Dedicated Advanced Centre (UMPEDAC). He is also a Distinguish Adjunct Professor at Renewable Energy Research Group, King Abdulaziz University, Jeddah, Saudi Arabia. His research interests include power electronics, Solar PV and wind technologies, real-time control systems, and electrical drives. Prof. N A Rahim is a fellow of the Institution of Engineering and Technology, U.K., the Academy of Sciences Malaysia and Senior Member IEEE. He is also a Chartered Engineer (U.K).