Pattanaik *et al.*
Journal of Electrical Systems and Inf Technol (2019) 6:1
https://doi.org/10.1186/s43067-019-0001-4

Journal of Electrical Systems and Information Technology

RESEARCH Open Access

Dynamic economic dispatch: a comparative study for differential evolution, particle swarm optimization, evolutionary programming, genetic algorithm, and simulated annealing

Jagat Kishore Pattanaik[1*], Mousumi Basu[1] and Deba Prasad Dash[2]

*Correspondence:
jagat.ju@gmail.com
[1] Department of Power Engineering, Jadavpur University, Salt Lake City 700098, Kolkata, India
Full list of author information is available at the end of the article

Abstract

This paper presents a comparative study for five artificial intelligent (AI) techniques to the dynamic economic dispatch problem: differential evolution, particle swarm optimization, evolutionary programming, genetic algorithm, and simulated annealing. Here, the optimal hourly generation schedule is determined. Dynamic economic dispatch determines the optimal scheduling of online generator outputs with predicted load demands over a certain period of time taking into consideration the ramp rate limits of the generators. The AI techniques for dynamic economic dispatch are evaluated against a ten-unit system with nonsmooth fuel cost function as a common testbed and the results are compared against each other.

Keywords: Dynamic economic dispatch, Differential evolution, Particle swarm optimization, Evolutionary programming, Genetic algorithm, Simulated annealing

Introduction

Static economic dispatch (SED) allocates the load demand which is constant for a given interval of time, among the online generators economically while satisfying various constraints including static behavior of the generators. Dynamic economic dispatch (DED) is an extension of static economic dispatch problem. DED is the most accurate formulation of the economic dispatch problem, but it is the most difficult to solve because of its large dimensionality. The first paper in this area appeared in 1972 [1] by Bechert and Kwatny. Since the DED was introduced, several methods [2–13] such as Lagrangian relaxation, gradient projection method, dynamic programming, hybrid EP and SQP, hybrid HNN-QP, hybrid differential evolution, etc., have been employed for solving this problem. However, all of these methods may not be able to find an optimal solution and usually stuck at a local optimum solution.

Recently, stochastic search algorithms [14–27] such as simulated annealing (SA), genetic algorithm (GA), evolutionary programming (EP), particle swarm optimization (PSO), and differential evolution (DE) have been successfully used to solve power system optimization problems due to their ability to find the near-global solution of a nonconvex optimization problem.

This paper investigates the applicability of the following five different AI techniques in dynamic economic dispatch (DED) problem: differential evolution (DE), particle swarm optimization (PSO), evolutionary programming (EP), genetic algorithm (GA), and simulated annealing (SA). The AI techniques are evaluated against a test system as a common testbed for comparison with each other.

Methods

Several artificial intelligence (AI) methods have evolved in recent past that facilitate to solve optimization problems which were previously difficult or impossible to solve. These techniques include differential evolution, particle swarm optimization, evolutionary programming, genetic algorithm, simulated annealing, etc. Reports of applications of each of these techniques have been widely published. The most important advantage of AI techniques lies in the fact that they are not limited by restrictive assumptions about the search space like continuity, existence of derivative of objective function, etc. These methods share some similarities. DE is introduced first, and followed by PSO, EP, GA, and SA.

Differential evolution

Differential evolution (DE) [14–16] is a type of evolutionary algorithm originally proposed by Price and Storn [14] for optimization problems over a continuous domain. DE is exceptionally simple, significantly faster, and robust. The basic idea of DE is to adapt the search during the evolutionary process. At the start of the evolution, the perturbations are large, since parent populations are far away from each other. As the evolutionary process matures, the population converges to a small region and the perturbations adaptively become small. As a result, the evolutionary algorithm performs a global exploratory search during the early stages of the evolutionary process and local exploitation during the mature stage of the search. In DE, the fittest of an offspring competes one-to-one with that of corresponding parent which is different from other evolutionary algorithms. This one-to-one competition gives rise to faster convergence rate. Price and Storn gave the working principle of DE with simple strategy in [14]. Later on, they suggested ten different strategies of DE [16]. Strategy-7 (DE/rad/1/bin) is the most successful and widely used strategy. The key parameters of control in DE are population size (N_P), scaling factor (S_F), and crossover constant (C_R). The optimization process in DE is carried out with three basic operations: mutation, crossover, and selection. The DE algorithm is described as follows:

Initialization

The initial population of N_P vectors is randomly selected based on uniform probability distribution for all variables to cover the entire search uniformly. Each individual X_i is a vector that contains as many parameters as the problem decision variables D. Random values are assigned to each decision parameter in every vector according to:

$$X_{ij}^0 \sim U\left(X_j^{\min}, X_j^{\max}\right), \tag{1}$$

where $i = 1, \ldots, N_P$ and $j = 1, \ldots, D$; $X_j^{\min}$ and $X_j^{\max}$ are the lower and upper bounds of the jth decision variable; $U\left(X_j^{\min}, X_j^{\max}\right)$ denotes a uniform random variable ranging

over $\left[X_j^{\min}, X_j^{\max}\right]$. X_{ij}^0 is the initial jth variable of ith population. All the vectors should satisfy the constraints. Evaluate the value of the cost function $f\left(X_i^0\right)$ of each vector.

Mutation

DE generates new parameter vectors by adding the weighted difference vector between two population members to a third member. For each target vector X_i^g at gth generation, the noisy vector $X_i^{/g}$ is obtained by

$$X_i^{/g} = X_a^g + S_F\left(X_b^g - X_c^g\right), \quad i \in N_P, \tag{2}$$

where X_a^g, X_b^g and X_c^g are selected randomly from N_P vectors at gth generation and $a \neq b \neq c \neq i$. The scaling factor (S_F), in the range $0 < S_F \leq 1.2$, controls the amount of perturbation added to the parent vector. The noisy vectors should satisfy the constraint.

Crossover

Perform crossover for each target vector X_i^g with its noisy vector $X_i^{/g}$ and create a trial vector $X_i^{//g}$, such that

$$X_i^{//g} = \begin{cases} X_i^{/g}, & \text{if } \rho \leq C_R \\ X_i^g, & \text{otherwise} \end{cases}, \quad i \in N_P, \tag{3}$$

where ρ is an uniformly distributed random number within [0, 1]. The crossover constant (C_R), in the range $0 \leq C_R \leq 1$, controls the diversity of the population and aids the algorithm to escape from local optima.

Selection

Perform selection for each target vector, X_i^g by comparing its cost with that of the trial vector, $X_i^{//g}$. The vector that has lesser cost of the two would survive for the next generation:

$$X_i^{g+1} = \begin{cases} X_i^{//g}, & \text{if } f\left(X_i^{//g}\right) \leq f\left(\mathrm{X}_i^g\right) \\ X_i^g, & \text{otherwise} \end{cases}, \quad i \in N_P \tag{4}$$

The process is repeated until the maximum number of generations or no improvement is seen in the best individual after many generations.

Particle swarm optimization

Particle swarm optimization (PSO) [17, 18] has been developed under the scope of artificial life where it is inspired by the natural phenomenon of fish schooling or birds flocking. PSO is basically based on the fact that in the quest of reaching the optimum solution in a multi-dimensional space, a population of particles is created whose present coordinate determines the cost function to be minimized. After each iteration, the new velocity and the new position of each particle are updated on the basis of a summated influence of each particle's present velocity, distance of the particle from its own best performance achieved so far during

the search process and the distance of the particle from the leading particle, i.e., the particle which at present is globally the best particle producing till now the best performance.

Usually, x and v are the variables employed to denote the position and the velocity of a particle in a multidimensional solution space. In a d dimensional space, the position and velocity of a particle i are represented as $d \times 1$ vectors, $x_i = (x_{i1}, x_{i2}, \ldots, x_{id})$ and $v_i = (v_{i1}, v_{i2}, \ldots, v_{id})$ respectively. For each particle i, the best position found so far is stored as another $d \times 1$ vector $p\text{best}_i = (p\text{best}_{i1}, p\text{best}_{i2}, \ldots, p\text{best}_{id})$. The best global particle among all particle i is denoted as $g$best and its coordinate in the dth dimension is given as $g\text{best}_d$. Hence, the velocity and position update equations for the ith particle in the dth dimension in the $(k+1)$th iteration, based on the performance in kth iteration are given as:

$$v_{id}^{(k+1)} = w \times v_{id}^{k} + c_1 \times rand() \times \left(p\text{best}_{id} - x_{id}^{k}\right) + c_2 \times rand() \times \left(g\text{best}_{d} - x_{id}^{k}\right) \tag{5}$$

$$x_{id}^{(k+1)} = x_{id}^{k} + v_{id}^{(k+1)}, \quad i \in N_P, \quad d \in D, \tag{6}$$

where D stands for the total number of dimensions for the multidimensional search problem and N_P stands for the population size. c_1 and c_2 give acceleration constants which provide relative stochastic weighting (implemented by the $rand()$ function which generates any value $\in [0, 1]$) of the deviation from the best own performance of the particle itself and the best performance of the group as a whole, so far, in the dth dimension. The velocity of the particle in the kth iteration in the dth dimension is given as $v_d^{\min} \leq v_{id}^{k} \leq v_d^{\max}$. Here, $v^{\max}$ is influential to determine the resolution with which regions are to be searched between the present position and the target position. A proper value should be chosen, such that $v^{\max}$ is neither too high nor too small.

The present system employs the PSO algorithm with adaptable inertia weight w, during the entire process of search, so that we can obtain a suitable balance between global and local explorations. In this work, the inertia weight w is set according to the following equation:

$$w = w_{\max} - \frac{w_{\max} - w_{\min}}{\text{iter}_{\max}} \times \text{iter}, \tag{7}$$

where $\text{iter}_{\max}$ is the maximum number of iterations and iter is the current number of iterations. We start with a high value of $w_{\max}$, such that we can perform aggressive global search initially in quest of potential good solution and gradually reduce w, such that we can fine tune our search locally as we move closer and closer to the minimum point.

Evolutionary programming

Evolutionary programming (EP) [20] is a technique in the field of evolutionary computation. It seeks the optimal solution by evolving a population of candidate solutions over a number of generations or iterations. During each iteration, a second new population is formed from an existing population through the use of a mutation operator. This operator produces a new solution by perturbing each component of an existing solution by a random amount. The degree of optimality of each of the candidate solutions or individuals is measured by their fitness, which can be defined as a function

of the objective function of the problem. Through the use of a competition scheme, the individuals in each population compete with each other. The winning individuals form a resultant population, which is regarded as the next generation. For optimization to occur, the competition scheme must be such that the more optimal solutions have a greater chance of survival than the poorer solutions. Through this, the population evolves towards the global optimal point. The algorithm is described as follows:

1. Initialization: The initial population of control variables is selected randomly from the set of uniformly distributed control variables ranging over their upper and lower limits. The fitness score f_i is obtained according to the objective function and the environment.
2. Statistics: The maximum fitness $f_{\max}$, minimum fitness $f_{\min}$, the sum of fitness $\sum f$, and average fitness f_{avg} of this generation are calculated.
3. Mutation: Each selected parent, for example, X_i, is mutated and added to its population with the following rule:

$$X_{i+m,j} = X_{ij} + N\left(0, \gamma\left(\overline{x}_j - \underline{x}_j\right)\frac{f_i}{f_{\max}}\right), \quad j \in D,\ i \in N_P, \tag{8}$$

where D is the number of decision variables in an individual, N_P is the population size, X_{ij} denotes the j th element of the ith individual; $N\left(\mu, \sigma^2\right)$ represents a Gaussian random variable with mean μ and variance σ^2; $f_{\max}$ is the maximum fitness of the old generation which is obtained in statistics; $\overline{x}_j$ and $\underline{x}_j$ are, respectively, maximum and minimum limits of the jth element; and γ is the mutation scale, $0 < \gamma \leq 1$, that could be adaptively decreased during generations. If any mutated value exceeds its limit, it will be given the limit value. The mutation process allows an individual with larger fitness to produce more offspring for the next generation.
4. Competition: Several individuals (k) which have the best fitness are kept as the parents for the next generation. Other individuals in the combined population of size ($2N_P - k$) have to compete with each other to get their chances for the next generation. A weight value w_i of the ith individual is calculated by the following competition:

$$w_i = \sum_{t=1}^{\mathrm{N}_t} w_{i,t}, \tag{9}$$

where N_t is the competition number generated randomly; $w_{i,t}$ is either 0 for loss or 1 for win as the ith individual competes with a randomly selected (rth) individual in the combined population. The value of $w_{i,t}$ is given in the following equation:

$$w_{i,t} = \begin{cases} 1 & \text{if } f_i < f_r \\ 0 & \text{otherwise,} \end{cases} \tag{10}$$

where f_r is the fitness of randomly selected rth individuals, and f_i is the fitness of the i th individual. When all $2N_P$ individuals, get their competition weights, they will be ranked in a descending order according to their corresponding value w_i. The first m individuals are selected along with their corresponding fitness f_i to be the bases for the next generation. The maximum, minimum, and the average fitness and the sum of the fitness of the current generation are then calculated in the statistics.

5. Convergence test: If the convergence condition is not met, the mutation and competition will run again. The maximum generation number can be used for convergence condition. Other criteria, such as the ratio of the average and the maximum fitness of the population, are computed and generations are repeated until

$$\left\{f_{\text{avg}}/f_{\max}\right\} \geq \delta, \tag{11}$$

where δ should be very close to 1, which represents the degree of satisfaction. If the convergence has reached a given accuracy, an optimal solution has been found for an optimization problem.

Genetic algorithm

Genetic algorithm [21] is based on the mechanics of natural selection. An initial population of candidate solutions is created randomly. Each of these candidate solutions is termed as individual. Each individual is assigned a fitness, which measures its quality. During each generation of the evolutionary process, individuals with higher fitness are favored and more probabilities to be selected as parents. After parents are selected for reproduction, they produce children via the processes of crossover and mutation. The individuals formed during reproduction explore different areas of the solution space. These new individuals replace lesser fit individuals of the existing population.

Due to difficulties of binary representation when dealing with continuous search space with large dimensions, the proposed approach has been implemented using real-coded genetic algorithm (RCGA) [22, 23]. The simulated Binary Crossover (SBX) and polynomial mutation are explained as follows.

Simulated binary crossover (SBX) operator

The procedure of computing child populations c_1 and c_2 from two parent populations y_1 and y_2 under SBX operator as follows:

1. Create a random number u between 0 and 1.
2. Find a parameter γ using a polynomial probability distribution as follows:

$$\gamma = \begin{cases} (u\gamma)^{1/(\eta_c+1)}, & \text{if } u \leq \frac{1}{\gamma} \\ \left(1/(2-u\gamma)\right)^{1/(\eta_c+)}, & \text{otherwise,} \end{cases} \tag{12}$$

where $\gamma = 2 - \delta^{-(\eta_c+1)}$. and $\delta = 1 + \frac{2}{y_2 - y_1}\min\left[\left(y_1 - y_l\right), \left(y_u - y_2\right)\right]$

Here, the parameter y is assumed to vary in $[y_l, y_u]$. Here, the parameter η_c is the distribution index for SBX and can take any non-negative value. A small value of η_c allows the creation of child populations far away from parents and a large value restricts only near-parent populations to be created as child populations.

3. The intermediate populations are calculated as follows:

$$\begin{aligned} c_{p1} &= 0.5\left[\left(y_1 + y_2\right) - \gamma\left(\left|y_2 - y_1\right|\right)\right] \\ c_{p2} &= 0.5\left[\left(y_1 + y_2\right) + \gamma\left(\left|y_2 - y_1\right|\right)\right] \end{aligned} \tag{13}$$

Each variable is chosen with a probability p_c and the above SBX operator is applied variable-by-variable.

Polynomial mutation operator

A polynomial probability distribution is used to create a child population in the vicinity of a parent population under the mutation operator. The following procedure is used:

1. Create a random number u between 0 and 1.
2. Calculate the parameter δ as follows:

$$\delta = \begin{cases} \left[2u + (1-2u)(1-\varphi)^{(\eta_m+1)}\right]^{\frac{1}{(\eta_m+1)}} - 1, & \text{if } u \le 0.5 \\ 1 - \left[2(1-u) + 2(u-0.5)(1-\varphi)^{(\eta_m+1)}\right]^{\frac{1}{(\eta_m+1)}}, & \text{otherwise,} \end{cases} \tag{14}$$

   where $\phi = \frac{\min[(c_p - y_l),(y_u - c_p)]}{(y_u - y_l)}$

   The parameter η_m is the distribution index for mutation and takes any non-negative value.
3. Calculate the mutated child as follows:

$$c_1 = c_{p1} + \delta(y_u - y_l)$$
$$c_2 = c_{p2} + \delta(y_u - y_l)$$

The perturbance in the population can be adjusted by varying η_m and p_m with generations as given below:

$$\eta_m = \eta_{m\min} + \text{gen} \tag{15}$$

$$p_m = \frac{1}{D} + \frac{\text{gen}}{\text{gen}_{\max}}\left(1 - \frac{1}{D}\right), \tag{16}$$

where $\eta_{m\min}$ is the user-defined minimum value for η_m, p_m is the probability of mutation, and D is the number of decision variables.

Simulated annealing

Simulated annealing [25, 26] is a powerful optimization technique which exploits the resemblance between a minimization process and the cooling of molten metal. The physical annealing process is simulated in the simulated annealing (SA) technique for the determination of global or near-global optimum solutions for optimization problems. In this algorithm, a parameter T_0, called temperature, is defined. Starting from a high temperature, a molten metal is cooled slowly until it is solidified at a low temperature. The iteration number in the SA technique is analogous to the temperature level. During each iteration, a candidate solution is generated. If this solution is a better solution, it will be accepted and used to generate yet another candidate solution. If it is a deteriorated solution, the solution will be accepted when its probability of acceptance $\Pr(\Delta)$ as given by Eq. (17) is greater than a randomly generated number between 0 and 1:

$$\Pr(\Delta) = 1/\left(1 + \exp\left(\Delta/T_v\right)\right), \tag{17}$$

where Δ is the amount of deterioration between the new and the current solutions, and T_v is the temperature at which the new solution is generated. Accepting deteriorated solutions in the above manner enables the SA solution to 'jump' out of the local optimum solution points and to seek the global optimum solution. In forming the new solution, the current solution is perturbed [28] according to the Gaussian probability distribution function (GPDF). The mean of the GPDF is taken to be the current solution, and its standard deviation is given by the product of the temperature and a scaling factor σ. The value of σ is less than one, and together with the value of temperature, it governs the size of the neighborhood space of the current solution and hence the amount of perturbation. The amount of perturbation is dependent upon the temperature when σ is kept at a constant value. In each iteration, the procedure for generating and testing the candidate solution is repeated for a specified number of trials, so that thermal equilibrium is reached for each temperature. The last accepted candidate solution is then taken as the starting solution for the generation of candidate solutions in the next iteration. Simulated annealing with a slow cooling schedule usually has larger capacity to find the optimal solution than that of a fast cooling schedule. The reduction of the temperature in successive iterations is governed by the following geometric function [25]:

$$T_v = r^{(v-1)} T_0, \tag{18}$$

where v is the iteration number and r is temperature reduction factor. T_0 is the initial temperature, the value of which can be set arbitrarily or estimated using the method described in Ref. [25]. The iterative process is terminated when there is no significant improvement in the solution after a prespecified number of iterations. It can also be terminated when the maximum number of iterations is reached.

Problem formulation

Normally, the DED problem minimizes the following total production cost of committed units:

$$F = \sum_{t=1}^{T} \sum_{i=1}^{N} F_{it}(P_{it}) \tag{19}$$

The fuel cost function of each unit considering valve-point effect [24] can be expressed as:

$$F_{it}(P_{it}) = a_i + b_i P_{it} + c_i P_{it}^2 + \left| d_i \sin\left(e_i\left(P_i^{\min} - P_{it}\right)\right)\right| \tag{20}$$

Subject to the following equality and inequality constraints for the tth interval in the scheduled horizon:

(i) Real power balance

$$\sum_{i=1}^{N} P_{it} - P_{Dt} - P_{Lt} = 0 \quad t \in T \tag{21}$$

(ii) Real power operating limits

$$P_i^{\min} \le P_{it} \le P_i^{\max} \quad i \in \mathrm{N}, \quad t \in \mathrm{T} \tag{22}$$

(iii) Generating unit ramp rate limits

$$\begin{aligned} P_{it} - P_{i(t-1)} &\le \mathrm{UR}_i, \quad i \in N, \quad t = 2, 3, \ldots, T \\ P_{i(t-1)} - P_{it} &\le \mathrm{DR}_i, \quad i \in N, \quad t = 2, 3, \ldots, T \end{aligned} \tag{23}$$

Determination of generation level of slack generator

N committed generators deliver their power output subject to the power balance constraint (21), the respective capacity constraints (22), and generating unit ramp rate limits (23). Assuming the power loading of first $(N-1)$ generators are known, the power level of the Nth generator (i.e., the slack generator) is given by:

$$P_{Nt} = P_{Dt} + P_{Lt} - \sum_{i=1}^{N-1} P_{it} \quad t \in T \tag{24}$$

The transmission loss P_{Lt} is a function of all the generators including that of the dependent generator and it is given by:

$$P_{Lt} = \sum_{i=1}^{N-1}\sum_{j=1}^{N-1} P_{it} B_{ij} P_{jt} + 2P_{\mathrm{N}t}\left(\sum_{i=1}^{N-1} B_{Ni} P_{it}\right) + B_{NN} P_{Nt}^2 \quad t \in T \tag{25}$$

Expanding and rearranging, Eq. (24) becomes:

$$B_{NN} P_{Nt}^2 + \left(2\sum_{i=1}^{N-1} B_{Ni} P_{it} - 1\right) P_{Nt} + \left(P_{Dt} + \sum_{i=1}^{N-1}\sum_{j=1}^{N-1} P_{it} B_{ij} P_{jt} - \sum_{i=1}^{N-1} P_{it}\right) = 0 \quad t \in T \tag{26}$$

The loading of the dependent generator (i.e., Nth) can then be found by solving Eq. (26) using standard algebraic method.

Results and discussion

A comparative study is performed for the five AI techniques for solving the dynamic economic dispatch (DED) problem for a ten-unit test system with nonsmooth fuel cost function is used. The demand of the system has been divided into 24 intervals. Unit data have been adopted from [10]. All AI techniques for the DED problem are implemented using MATLAB 7.0 on a PC (Pentium-IV, 3.0 GHz). The DED problem is solved by using DE, PSO, EP, RCGA, and SA. In case of DE, the population size (N_{P}), scaling factor (S_F), and crossover rate (C_R) have been selected as 50, 0.75, and 1.0, respectively, for the test system under consideration. In case of PSO, parameters are taken as $N_{\mathrm{P}} = 50$, $w_{\max} = 0.2$, $w_{\min} = 0.05$, $c_1 = 0.35$, and $c_2 = 0.35$. The population size (N_{P}) and scaling factor (F) have been selected as 100 and 0.1, respectively, in case of EP. In case of RCGA, the population size, crossover, and mutation probabilities have been selected as 100, 0.07, and 0.5, respectively. The initial temperature (T_0) of SA algorithm has been determined using the procedures described in [28]. As per guideline [25], the value of r lies in the range from 0.80 to 0.99. For seeking the optimal solution, the value of r is required to be set close to 0.99, so that a slow cooling process is simulated. The appropriate setting of r is set by experimenting with its value in the range from 0.95 to 0.99, and this value is found to be 0.98. The number of trials at each temperature has been taken 30. Maximum number of generations has

Table 1 Hourly generation (MW) schedule, cost ($\times 10^6$\$), and CPU time (second) obtained from dynamic economic dispatch from DE

Hour	P_1	P_2	P_3	P_4	P_5	P_6	P_7	P_8	P_9	P_{10}	Cost	CPU time
1	150.0023	135.0000	99.3971	130.0884	134.8637	130.7406	93.3093	119.9992	52.0598	10.0005	2.5003	45.68
2	150.0041	151.4090	98.8454	173.8606	173.0558	147.6245	85.2620	119.9991	22.2828	10.0202		
3	150.0049	135.0059	177.4829	181.4962	184.7015	142.3316	103.1999	119.9994	52.1827	40.0179		
4	150.0000	198.1133	235.2510	192.4256	168.5245	155.7928	129.5929	119.9994	48.8509	43.4193		
5	199.5846	194.9589	228.6988	232.4826	173.9633	145.4818	129.6382	119.9993	52.0771	43.4303		
6	204.3037	214.7293	256.7306	244.9424	223.0072	159.9992	129.7790	119.9997	79.9974	43.4224		
7	196.2184	215.7100	330.8211	253.0177	230.3517	156.3752	129.7308	119.9974	79.9926	43.4255		
8	203.8212	221.1860	338.4768	299.9994	237.9047	159.9978	129.7447	119.9994	79.9977	43.4287		
9	267.0289	299.6440	339.9952	299.9993	242.9981	159.9981	129.8928	119.9998	79.9956	54.9987		
10	311.4126	362.1257	340.0000	299.9998	242.9991	160.0000	129.9997	120.0000	79.9995	54.9998		
11	368.7806	397.0834	339.9981	300.0000	242.9996	159.9988	129.9999	120.0000	79.9999	54.9994		
12	359.9575	454.5160	340.0000	299.9993	243.0000	159.9998	129.9967	119.9999	79.9992	54.9985		
13	344.5213	383.8998	339.9993	300.0000	242.9992	159.9999	129.9996	119.9989	80.0000	54.9999		
14	279.9978	305.5589	337.4480	299.9571	242.9963	156.6715	129.9970	119.9987	78.7665	43.4217		
15	218.3373	238.0725	300.2687	299.9968	242.9983	160.0000	129.9992	119.9994	80.0000	45.0862		
16	150.0060	173.0080	232.6009	270.1371	240.8416	159.6082	129.7414	119.9973	78.3523	43.4221		
17	199.5903	135.0895	225.9981	244.4318	230.8371	136.8053	129.7877	119.9996	53.9852	43.4227		
18	218.2552	201.3210	212.2215	292.9677	233.1928	159.9959	129.9329	119.9995	65.6852	43.4266		
19	222.2039	245.1578	291.0885	300.0000	242.9921	159.9998	129.9972	119.9998	80.0000	43.4184		
20	295.3769	323.7551	339.8164	299.9942	242.9711	159.9945	129.9967	119.9983	79.9978	54.9604		
21	240.4714	326.1261	339.9993	300.0000	243.0000	159.9981	129.9946	120.0000	80.0000	54.9986		
22	160.4729	246.1266	288.0628	260.6882	233.1695	159.3605	129.7499	103.8490	52.0353	43.4261		
23	150.0000	169.8554	210.8364	238.6813	187.8379	122.4697	107.5329	85.3143	48.3700	43.4208		
24	150.0009	135.0020	132.5246	215.2546	174.0883	84.9670	129.6527	93.0678	51.5500	43.4249		

Table 2 Hourly generation (MW) schedule, cost (× 10^6\$), and CPU time (second) of dynamic economic dispatch obtained from PSO

Hour	P_1	P_2	P_3	P_4	P_5	P_6	P_7	P_8	P_9	P_{10}	Cost	CPU time
1	150.0000	199.7927	74.4886	88.3077	145.0100	124.9606	96.7047	56.0175	77.5430	43.4224	2.5484	46.84
2	150.0017	147.5373	148.7770	149.5742	137.8931	124.4079	93.2824	85.4043	52.0624	43.4216		
3	150.0044	213.0136	199.1157	182.6784	142.3358	129.7013	94.0599	85.3409	47.5164	43.4298		
4	187.3955	235.1569	227.6323	192.4983	182.8953	129.9500	103.0765	88.8075	52.0588	43.4274		
5	178.0093	294.7498	217.7913	127.9445	224.1009	159.9976	129.9779	92.2447	53.2028	43.4213		
6	223.6870	281.8095	227.5416	185.1988	242.9956	159.9939	129.9922	119.9993	63.3236	43.4223		
7	217.3248	297.5720	297.3497	189.3693	222.7063	159.9881	129.7828	119.9955	79.3220	43.4204		
8	232.9554	319.3586	339.9955	156.2502	242.9997	159.9959	129.9955	120.0000	79.9994	54.9958		
9	285.8364	367.8710	339.9130	214.5776	242.9967	159.9962	129.8884	120.0000	79.9998	54.9938		
10	324.7413	375.6777	339.8941	274.3362	242.4056	159.9982	129.9994	119.9990	79.9977	54.9763		
11	369.0578	396.8040	339.9989	300.0000	243.0000	159.9997	129.9999	120.0000	80.0000	54.9998		
12	352.8360	461.6681	339.9983	299.9991	242.9987	159.9990	129.9974	119.9991	79.9992	54.9973		
13	355.8826	372.5578	339.9966	300.0000	243.0000	159.9996	129.9990	120.0000	79.9993	54.9994		
14	271.5405	295.9305	339.9964	299.8609	242.9189	159.9943	129.9107	119.9985	79.9938	54.4185		
15	211.3561	229.5151	316.4836	299.9914	242.9988	159.9985	129.9899	119.9992	79.9988	44.3025		
16	200.6578	221.9566	230.3878	236.6025	200.0030	159.9962	129.6434	120.0000	55.9055	43.4237		
17	195.2612	199.3270	219.4508	200.6222	203.9711	131.0978	127.2571	119.9996	79.9992	43.4184		
18	193.4743	236.3698	201.0210	279.8192	233.1585	159.9989	129.7883	119.9985	80.0000	43.4300		
19	176.0778	286.0752	296.7001	299.9717	242.9584	159.9676	129.7458	119.9992	79.9988	43.4211		
20	247.2880	371.7111	339.9992	299.9992	242.9959	159.9988	129.9995	119.9985	79.9967	54.9923		
21	257.8218	325.2963	339.8566	291.6268	242.9982	159.9995	129.9989	119.9994	79.9996	47.1784		
22	224.7270	236.9661	246.2622	242.6429	216.6146	159.9996	129.8726	119.9998	56.8388	43.4247		
23	150.0007	149.7524	258.0390	180.7417	156.7223	157.4447	128.8627	90.0082	49.0953	43.4169		
24	191.6432	135.0016	177.4172	157.5173	128.9581	132.6815	104.8532	86.0083	52.2116	43.4223		

Table 3 Hourly generation (MW) schedule, cost (× 10^6\$), and CPU time (second) of dynamic economic dispatch obtained from EP

Hour	P_1	P_2	P_3	P_4	P_5	P_6	P_7	P_8	P_9	P_{10}	Cost	CPU time
1	178.2974	152.4782	175.9679	152.2693	82.6967	100.2570	55.1840	86.1211	32.0525	40.7531	2.5722	68.47
2	151.9730	135.0000	213.2147	201.7864	131.9888	118.5195	33.4036	82.8231	35.5465	28.5607		
3	203.1254	163.9726	251.9514	199.1529	138.8356	113.3658	42.5714	70.8768	56.7945	47.1242		
4	150.7934	200.2340	325.5753	226.0880	162.2524	113.6450	58.7423	96.9791	62.9590	45.6968		
5	173.3115	144.6134	285.1757	275.9783	211.5299	135.3282	86.1134	107.2580	52.8395	47.9709		
6	193.0859	180.9466	295.7199	294.4164	243.0000	150.2837	91.6984	119.8831	59.2527	48.5979		
7	199.5730	227.0251	317.7981	285.0641	232.0515	149.8511	121.3454	95.4300	74.6611	53.1300		
8	230.4454	246.3501	318.5516	281.4019	243.0000	160.0000	130.0000	118.9990	55.1984	50.9417		
9	309.2169	325.9173	340.0000	300.0000	240.8720	159.7176	127.3408	107.4298	34.9745	50.1778		
10	319.3131	405.6572	338.3825	291.2641	243.0000	155.7018	130.0000	120.0000	54.6931	44.5437		
11	376.5109	470.0000	293.8462	299.8919	241.4655	145.5909	128.4700	117.7083	76.6091	45.5225		
12	410.5210	460.0330	330.1543	290.0606	240.5785	157.8225	130.0000	99.3619	72.5975	52.6241		
13	354.5582	458.7721	325.3415	294.7094	243.0000	158.6978	125.2569	100.0073	61.8662	35.9623		
14	300.6076	379.5915	301.5840	285.1112	215.5048	148.7961	130.0000	119.2818	71.6677	43.9232		
15	229.4357	300.8396	328.6496	276.8651	236.1853	136.8260	118.9267	102.5443	68.2499	37.5047		
16	150.0000	230.9564	290.8325	285.5966	226.2529	129.9050	111.5957	95.6673	39.5550	38.3823		
17	168.5816	155.4401	312.8637	259.7161	197.5561	142.4882	119.8778	84.3126	39.2104	40.0607		
18	150.0000	235.2465	326.6851	295.5718	233.0072	131.9463	121.1258	103.6587	43.9754	36.0901		
19	225.5372	252.2426	308.9457	293.8158	241.5050	160.0000	130.0000	120.0000	72.9211	29.9004		
20	304.6380	331.5404	337.4662	300.0000	243.0000	154.7102	127.7088	118.4984	80.0000	49.6164		
21	290.6766	310.6099	340.0000	299.9669	242.0460	159.8559	130.0000	117.6984	72.4081	32.2045		
22	211.5595	231.5171	260.5104	288.5803	242.1627	130.7513	110.3033	101.3453	55.6514	45.1916		
23	150.0000	165.9392	212.1930	239.0283	196.6465	98.0630	116.2304	119.0883	29.2393	37.8554		
24	180.3674	135.0000	133.2234	190.3317	168.2021	132.2441	104.0532	98.6428	54.6286	12.8466		

Table 4 Hourly generation (MW) schedule, cost (× $10^6$$), and CPU time (second) of dynamic economic dispatch obtained from RCGA

Hour	P_1	P_2	P_3	P_4	P_5	P_6	P_7	P_8	P_9	P_{10}	Cost	CPU time
1	150.0000	137.3240	165.8277	130.9594	185.8528	86.6659	43.1268	82.1384	60.5076	13.3651	2.5854	72.68
2	154.0813	142.0823	111.6599	173.9694	212.5123	127.5569	26.5848	90.0247	58.4067	35.8485		
3	167.6789	175.6266	190.4018	182.6740	243.0000	100.9178	33.9537	104.2804	43.8270	45.1084		
4	151.0869	234.2356	201.1268	192.8167	232.0634	140.5203	58.3984	120.0000	61.3047	51.1215		
5	179.0793	167.5926	241.0310	242.2296	233.1403	132.5917	85.7099	110.1910	76.4297	52.2067		
6	178.1383	246.7190	319.0410	243.9511	222.8359	130.1020	113.9272	114.8017	69.8714	38.1359		
7	231.2156	232.1208	330.9049	293.7810	238.4605	127.7300	101.5284	116.2500	70.6214	13.9564		
8	186.8206	299.0632	336.2252	278.7172	243.0000	152.4734	117.8664	115.2754	79.4536	26.5290		
9	254.2367	344.7199	340.0000	296.9019	235.7257	157.4128	125.5643	120.0000	72.5853	47.9489		
10	323.6424	423.9759	339.7272	282.1438	243.0000	157.9184	128.9391	97.5989	54.7205	51.3773		
11	379.9329	470.0000	333.0634	300.0000	236.4433	160.0000	115.5534	93.1074	54.6007	53.1176		
12	413.1634	443.5430	337.5237	289.1344	238.2085	159.5075	124.2170	119.3562	80.0000	38.7139		
13	335.4767	470.0000	334.4274	295.0397	230.2758	136.9080	129.4886	116.0717	73.7319	36.8421		
14	279.7477	390.3287	309.3307	277.8978	236.6148	157.8536	129.2263	93.2559	74.6412	47.1851		
15	225.8492	300.6198	340.0000	271.4831	235.7474	160.0000	113.9795	97.3194	60.2646	30.6238		
16	156.8943	221.9919	274.1830	222.1209	224.4294	145.5529	120.9893	119.0116	71.2232	41.9545		
17	150.0000	142.9721	307.5557	236.0399	229.9259	119.3528	128.4166	92.4584	77.2718	35.9486		
18	213.3822	209.6171	280.5045	285.5809	236.6284	160.0000	118.0363	89.8209	48.4504	35.0462		
19	216.8980	288.5752	330.2142	300.0000	233.9977	126.9242	128.6758	99.3605	75.2536	35.9281		
20	271.3901	367.8558	340.0000	298.0471	243.0000	160.0000	129.7412	108.7347	78.6677	49.8348		
21	285.6118	378.6250	327.2155	300.0000	240.5381	151.3773	127.6958	102.3417	51.2099	31.5443		
22	216.5251	289.8412	318.2870	250.3409	190.6218	112.0639	127.3546	100.7594	46.4267	26.1429		
23	157.4794	210.8063	256.3831	215.0895	181.2377	116.3920	97.4503	82.3338	59.7178	43.7112		
24	150.0000	140.8972	179.9885	221.9082	180.1364	106.8230	87.4166	63.2514	33.6222	45.5223		

Table 5 Hourly generation (MW) schedule, cost ($\times 10^6$\$), and CPU time (second) of dynamic economic dispatch obtained from SA

Hour	P_1	P_2	P_3	P_4	P_5	P_6	P_7	P_8	P_9	P_{10}	Cost	CPU time
1	150.0024	135.0015	144.9656	182.0074	88.6591	117.4978	93.2986	48.7102	52.0561	43.4221	2.5372	47.87
2	169.2861	164.0750	95.6618	242.5084	74.7126	124.2574	97.5492	67.7461	53.8366	43.4206		
3	184.8799	135.0007	155.2365	262.0329	128.0534	133.9343	102.1585	89.3892	52.9392	43.4240		
4	150.0000	181.9025	190.9163	298.3203	176.5955	123.4629	104.0068	118.5219	55.0922	43.4206		
5	150.0070	278.9117	211.4438	255.6995	173.2830	159.9991	96.0890	94.9458	57.1100	43.4203		
6	214.2334	219.2741	234.2596	299.9345	223.4172	159.5642	109.2608	119.9991	53.8481	43.4189		
7	175.2140	302.2295	288.1683	257.8746	199.9232	159.6388	129.8786	119.9980	79.9985	43.4209		
8	193.7402	249.1671	328.3684	299.9997	230.1077	159.9989	129.9220	119.9937	79.9983	43.4205		
9	269.6107	308.3153	339.9999	299.9989	242.9961	159.9990	129.9982	119.9994	80.0000	43.7628		
10	304.7557	368.7936	339.9997	300.0000	243.0000	159.9985	129.9982	120.0000	79.9998	54.9969		
11	370.9456	394.9326	339.9988	299.9998	242.9958	159.9992	129.9958	119.9987	79.9995	54.9979		
12	415.0976	399.4280	339.9995	300.0000	242.9998	159.9983	130.0000	120.0000	79.9996	54.9994		
13	329.9181	398.5121	340.0000	299.9998	243.0000	159.9997	129.9991	119.9986	79.9993	54.9975		
14	244.2169	353.4629	338.1337	281.4461	234.8856	159.9936	129.9123	119.9990	79.9969	53.0353		
15	224.1848	271.9778	312.9167	249.9984	242.9997	159.9848	129.7650	119.9959	80.0000	43.4258		
16	208.2949	213.3178	244.3962	189.9126	230.2628	154.8714	129.6713	104.5874	79.9986	43.4202		
17	204.6527	271.5673	198.7677	144.9524	195.0257	155.7953	129.9000	120.0000	57.1798	43.4220		
18	223.1500	216.8604	290.5531	181.9823	241.3903	154.5984	129.8428	119.9995	75.6032	43.4213		
19	227.0066	305.9172	311.0236	252.0665	206.4595	159.9960	129.8893	119.9986	79.9985	43.4214		
20	252.5758	366.3819	339.9997	299.9983	242.9980	159.9995	129.9989	120.0000	79.9992	54.9982		
21	242.5096	336.0323	328.2196	299.9964	242.9996	159.9964	129.9980	119.9997	79.9996	54.9789		
22	216.1082	237.8512	240.1953	241.5349	236.1456	159.9079	129.9959	119.9972	52.0808	43.4313		
23	150.0000	192.0533	182.0508	186.9352	176.1480	142.6721	129.0641	119.9987	41.9049	43.4173		
24	150.0012	218.1755	88.9043	129.8650	155.4283	123.1737	129.0892	119.9951	52.0545	43.4145		

Table 6 Comparison of cost and CPU time among five AI techniques

Method	Cost ($\times 10^6$\$)	CPU time (s)
DE	2.5003	45.68
PSO	2.5484	46.84
EP	2.5722	68.47
RCGA	2.5854	72.68
SA	2.5372	47.87

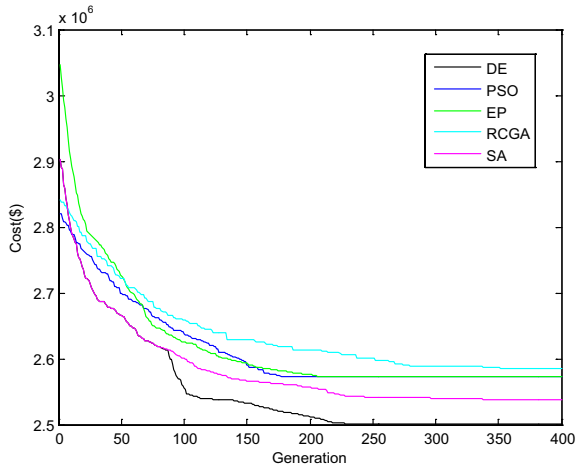

Fig. 1 Cost convergence characteristics obtained from different artificial intelligence techniques

been selected 400 for all the five AI techniques discussed in this paper. Tables 1, 2, 3, 4, 5 reveal hourly generation schedule, minimum production cost, and CPU time obtained from DE, PSO, EP, RCGA, and SA, respectively. Table 6 shows the comparison of cost and CPU time among DE, PSO, EP, RCGA, and SA. Figure 1 shows the cost convergence characteristics obtained from DE, PSO, EP, RCGA, and SA.

The present article describes different AI methods and applied on ten-unit test system. The cost convergence characteristics are obtained by the application of various AI method and its revealed that DE gives better result than others. The comparison can be done by the application of improved real-coded genetic algorithm (IRCGA) to dynamic economic dispatch problem. IRCGA will give better result than other AI methods defined in this article.

Conclusion

In this paper, artificial intelligent techniques have been applied to solve dynamic economic dispatch problem with nonsmooth fuel cost function. The results of the dynamic economic dispatch using different artificial intelligent techniques are almost identical. When the results are compared with each other, differential evolution seems to be better considering cost and CPU time.

List of symbols

Variables
P_{it}: real power output of i th unit during time interval t; $P_i^{\min}$, $P_i^{\max}$: lower and upper generation limits for i th unit; P_{Dt}: load demand at the time interval t; P_{Lt}: transmission line losses at time t; a_i, b_i, c_i, d_i, e_i: cost coefficients of i th unit; $F_{it}(P_{it})$: cost of producing real power output P_{it} at time t; UR_i, DR_i: ramp-up and ramp-down rate limits of the ith generator; N: number of generating units; N_P: population size; N_c: number of clones; T: number of intervals in the scheduled horizon.

Abbreviations
AI: artificial intelligence; DE: differential evolution; PSO: particle swarm optimization; SA: simulated annealing; DED: dynamic economic dispatch; SED: static economic dispatch; RCGA: real-coded genetic algorithm; IRCGA: improved real-coded genetic algorithm.

Acknowledgements
Not applicable.

Authors' contributions
JKP carried out the literature review, drafting of the paper, comparison of test results analysis for dynamic economic dispatch, differential evolution, particle swarm optimization methods and involved in simulation part for all five AI methods. MB involved in analysis for evolutionary programming, genetic algorithm techniques. DPD involved in analysis for simulated annealing and review different results obtained by various AI methods. All authors read and approved the final manuscript.

Authors' information
JKP received the M.E. degree in year 2011 and Ph.D degree in year 2019 in Electrical Engineering from Jadavpur University, Kolkata, India. The research interest of JKP is soft computing application in optimization of modern power system. JKP published more than ten research papers in international journals and conferences.

MB received the Ph.D. degree from Jadavpur University, Kolkata, India. MB currently working as a Professor in the department of Power Engineering, Jadavpur University. The research interests of MB are power system optimization and soft computing technique. MB is having a teaching experience of more than 25 years and has published several research papers in reputed international journals and conferences.

DPD received the Ph.D. degree from Jadavpur University, Kolkata, India. DPD currently working as an Associate Professor in the department of electrical engineering, Government College of Engineering, Kalahandi, Odisha. DPD has more than 15 years of teaching experience and research interests are power system stability and soft computing technique.

Funding
There is no funding information available for this manuscript.

Availability of data and materials
All data generated or analysed during this study are included in this published article.

Competing interests
The authors declare that they have no competing interests.

Author details
[1] Department of Power Engineering, Jadavpur University, Salt Lake City 700098, Kolkata, India. [2] Department of Electrical Engineering, Government College of Engineering, Kalahandi, Odisha, India.

Appendix

See Tables 7 and 8.

The transmission loss formula coefficients are:

Table 7 Generator characteristics

Unit	P_i^{max} MW	P_i^{min} MW	a_i $/h	b_i $/MWh	c_i $/(MW)^2$ h	d_i $/h	e_i rad/MW	UR_i MW/h	DR_i MW/h
1	150	470	786.7988	38.5397	0.1524	450	0.041	80	80
2	135	470	451.3251	46.1591	0.1058	600	0.036	80	80
3	73	340	1049.9977	40.3965	0.0280	320	0.028	80	80
4	60	300	1243.5311	38.3055	0.0354	260	0.052	50	50
5	73	243	1658.5696	36.3278	0.0211	280	0.063	50	50
6	57	160	1356.6592	38.2704	0.0179	310	0.048	50	50
7	20	130	1450.7045	36.5104	0.0121	300	0.086	30	30
8	47	120	1450.7045	36.5104	0.0121	340	0.082	30	30
9	20	80	1455.6056	39.5804	0.1090	270	0.098	30	30
10	10	55	1469.4026	40.5407	0.1295	380	0.094	30	30

Table 8 Load demands

Hour	P_D (MW)	Hour	P_D (MW)	Hour	P_D (MW)
1	1036	9	1924	17	1480
2	1110	10	2022	18	1628
3	1258	11	2106	19	1776
4	1406	12	2150	20	1972
5	1480	13	2072	21	1924
6	1628	14	1924	22	1628
7	1702	15	1776	23	1332
8	1776	16	1554	24	1184

$$B = \begin{bmatrix} 0.000049 & 0.000014 & 0.000015 & 0.000015 & 0.000016 & 0.000017 & 0.000017 & 0.000018 & 0.000019 & 0.000020 \\ 0.000014 & 0.000045 & 0.000016 & 0.000016 & 0.000017 & 0.000015 & 0.000015 & 0.000016 & 0.000018 & 0.000018 \\ 0.000015 & 0.000016 & 0.000039 & 0.000010 & 0.000012 & 0.000012 & 0.000014 & 0.000014 & 0.000016 & 0.000016 \\ 0.000015 & 0.000016 & 0.000010 & 0.000040 & 0.000014 & 0.000010 & 0.000011 & 0.000012 & 0.000014 & 0.000015 \\ 0.000016 & 0.000017 & 0.000012 & 0.000014 & 0.000035 & 0.000011 & 0.000013 & 0.000013 & 0.000015 & 0.000016 \\ 0.000017 & 0.000015 & 0.000012 & 0.000010 & 0.000011 & 0.000036 & 0.000012 & 0.000012 & 0.000014 & 0.000015 \\ 0.000017 & 0.000015 & 0.000014 & 0.000011 & 0.000013 & 0.000012 & 0.000038 & 0.000016 & 0.000016 & 0.000018 \\ 0.000018 & 0.000016 & 0.000014 & 0.000012 & 0.000013 & 0.000012 & 0.000016 & 0.000040 & 0.000015 & 0.000016 \\ 0.000019 & 0.000018 & 0.000016 & 0.000014 & 0.000015 & 0.000014 & 0.000016 & 0.000015 & 0.000042 & 0.000019 \\ 0.000020 & 0.000018 & 0.000016 & 0.000015 & 0.000016 & 0.000015 & 0.000018 & 0.000016 & 0.000019 & 0.000044 \end{bmatrix}$$

Received: 26 August 2019 Accepted: 15 October 2019
Published online: 27 November 2019

References

1. Bechert TE, Kwatny HG (1972) On the optimal dynamic dispatch of real power. IEEE Trans Power Appar Syst PAS-91:889–898
2. Ross DW, Kim S (1980) Dynamic economic dispatch of generation. IEEE Trans Power Appar Syst PAS-99(6):2060–2068
3. Wood WG (1982) Spinning reserve constrained static and dynamic economic dispatch. IEEE Trans Power Appar Syst PAS:381
4. Van Den Bosch PPJ (1985) Optimal dynamic dispatch owing to spinning-reserve and power-rate limits. IEEE Trans Power Appar Syst PAS-104(12):3395–3401

5. Granelli GP, Marannino P, Montagna M, Silvestri A (1989) Fast and efficient gradient projection algorithm for dynamic generation dispatching. IEE Proc Gen Transmiss Distrib 136(5):295–302
6. Hindi KS, AbGhani MR (1991) Dynamic economic dispatch for large scale power systems; a Lagrangian relaxation approach. Elect Power Syst Res 13(1):51–56
7. Lee FN, Lemonidis L, Liu K-C (1994) Price-based ramp-rate model for dynamic dispatch and unit commitment. IEEE Trans Power Syst 9(3):1233–1242
8. Travers DL, Kaye RJ (1998) Dynamic dispatch by constructive dynamic programming. IEEE Trans Power Syst 13:1
9. Han XS, Gooi HB, Kirschen DS (2001) Dynamic economic dispatch: feasible and optimal solutions. IEEE Trans Power Syst 16(1):22–28
10. Attavriyanupp P, Kita H, Tanaka T, Hasegawa J (2002) A hybrid EP and SQP for dynamic economic dispatch with nonsmooth fuel cost function. IEEE Trans Power Syst 17(2):411–416
11. Victoire TAA, Jeyakumar AE (2005) Reserve constrained dynamic dispatch of units with valve-point effects. IEEE Trans Power Syst 20(3):1273–1282
12. Abdelaziz AY, Kamh MZ, Mekhamer SF, Badr MAL (2008) A hybrid HNN-QP approach for dynamic economic dispatch problem. Electric Power Syst Res 78:1784–1788
13. Yuan Xiaohui, Wang Liang, Zhang Yongchuan, Yuan Yanbin (2009) A hybrid differential evolution method for dynamic economic dispatch with valve-point effects. Expert Syst Appl 36:4042–4048
14. Storn R, Price KV (1997) Differential evolution—a simple and efficient heuristic for global optimization over continuous spaces. J Global Optim 11(4):341–359
15. Lampinen J (2002) A constraint handling approach for the differential evolution algorithm. Proc Congr Evol Comput 2:1468–1473
16. Price KV, Storn R, Lampinen J (2005) Differential evolution: a practical approach to global optimization. Springer, Berlin
17. Kennedy J, Eberhart R (1995) Particle swarm optimization. Proc IEEE Int Conf Neural Netw 4:1942–1948
18. Shi Y, Eberhart R (1998) A modified particle swarm optimizer. Proc IEEE Int Conf Evol Comput 69:73
19. Gaing Zwe-Lee (2003) Particle swarm optimization to solving the economic dispatch considering the generator constraints. IEEE Trans PWRS 18(3):1187–1195
20. Yang HT, Yang PC, Huang CL (1996) Evolutionary Programming based economic dispatch for units with non-smooth fuel cost functions. IEEE Trans PWRS 11(1):112–118
21. Goldberg D (1989) Genetic algorithms in search, optimization & machine learning, reading. Addison-Wesley Publishing Company Inc, Boston
22. Deb K, Agrawal RB (1995) Simulated binary crossover for continuous search space. Complex Syst 9(2):115–148
23. Herrera F, Lozano M, Verdegay JL (1998) Tackling real-coded genetic algorithms: operators and tools for behavioral analysis. Artif Intell Rev 12(4):265–319
24. Walter DC, Sheble GB (1993) Genetic algorithm solution of economic dispatch with valve point loading. IEEE Trans Power Syst 8:1325–1332
25. Kirkpatrick S, Gelatt C, Vecchi M (1983) Optimization by simulated annealing. Science 22:671–680
26. Aarts E, Korst JM (1989) Simulated annealing and Boltzmann machines: a stochastic approach to combinatorial optimization and neural computing. John Wiley, New York
27. Laarhoven PJMV, Arts EHL (1987) Simulated annealing: theory and applications. D. Reidel, Dordrecht
28. Wong KP, Fung CC (1993) Simulated annealing based economic dispatch algorithm. IEE Proc Gen Transmiss Distrib 140(6):509–515

Publisher's Note