Improved Jaya Algorithm for Economic Dispatch Considering Valve-Point Effect and Multi-Fuel Options

CHUANHU CHEN1, DEXUAN ZOU1, AND CHUNLIANG LI2
1School of Electrical Engineering and Automation, Jiangsu Normal University, Xuzhou 221116, China
2Xuzhou Boyuan Technology Company Ltd., Xuzhou 221008, China
Corresponding author: Chuanhu Chen (cch@jsnu.edu.cn)

This work was supported by the Transformation Project of Xuzhou Science and Technology under Grant KC19234.

ABSTRACT In this work, an improved Jaya (IJaya) algorithm is proposed and used to solve the economic load dispatch (ELD) problem considering valve-point effect and multi-fuel options. Compared with original Jaya, the convergence rate of IJaya is accelerated by distance-varying acceleration coefficient (DVAC), and the population diversity is enlarged by a self-adaptive population mechanism (SAPM). Meanwhile, to avoid premature problem, Gaussian and Cauchy mutation (GCM) is used to help IJaya jump out of local optima. The superiorities of IJaya have been validated by solving ELD cases with 6, 10, 13, 15, 40 generators, and the results are analysed in aspects of convergence rate, solution quality and robustness. The conclusion is that, in the tested ELD cases, IJaya behaves much more competitive than Jaya under the same testing conditions.

INDEX TERMS Heuristic algorithm, improved jaya, optimization, economic dispatch.

I. INTRODUCTION Economic load dispatch (ELD) in power system is a high dimensional, non-convex, non-smooth and non-linear optimization problem under various of constraints. The main task of ELD problem is to allocate the working generators to fulfill the load demand requirement with minimum total operating cost, as well as to satisfy all the constraints of physical and operational limits of the power system [1]. Over the years, various of nature inspired meta-heuristic optimization approaches have been developed to address the ELD problem, such as genetic algorithm (GA) [2], particle swarm optimization (PSO) [3], biogeography-based optimization (BBO) [4], artificial immune system (AIS) [5], firefly algorithm [6], harmony search (HS) [7], artificial bee colony algorithm (ABC) [8] and teaching-learning-based optimization (TLBO) [9]. However, even though these heuristic methods obtained promising results in convergence rate and cost savings in solving ELD problems, they are easily be trapped by local optima rather than achieving global optima, which is an unavoidable drawback for heuristic methods.

In order to overcome the drawback, new optimization methods with modifications of existing approaches have been made to further search for global optima, such as chaos mutation firefly algorithm (CMFA) [10], new particle swarm optimization with local random search (NPSO-LRS) [11], DE with chaos sequences and sequential quadratic programming (DEC-SQP) [12], across neighborhood search algorithm with variable reduction strategy (ANS-VRS) [13]. Over the latest years, [14] proposed a phasor particle swarm optimization (PPSO) to solve different type of ELDs by substituting the control parameters of PSO with a phasor angle (\(\theta\)). In [15], a distributed pattern search algorithm (DPSA) is developed for ELD in smart grid by integrating a flooding-based topology discovery algorithm (FBTDA) under multi-agent systems. In [16], a modified crow search algorithm (MCSA) by innovative crows selection and adaptive adjustment of the flight length is proposed for solving the ELD problem. In [17], an orthogonal learning competitive swarm optimizer (OLCSO) combining with orthogonal learning is proposed to solve the ELD problem. However, the modifications or combinations of existing approaches lead to increased number of parameters, and the tuning of parameters is a very difficult task. Hence, an advanced method with fewer number of parameters is required.

In [18], a simple yet advanced heuristic algorithm named Jaya was proposed. Different from most of the other heuristic
algorithms requiring for algorithm-specific parameters, Jaya algorithm is totally free from algorithm-specific parameters and only two common parameters are required, which are population size \( pop \) and maximum number of iterations \( max\_iter \). This competitive benefit makes it popularly applied in various engineering applications [19], [20]. In [21], a novel multi-population based chaotic Jaya algorithm is proposed in solving ELD problem. In [22], Jaya algorithm with self-adaptive multi-population and Levy flights was used for solving the same problem. In [23], optimal design of heat exchangers was solved by elitist-Jaya algorithm. In [24], an improved Jaya was used for parameter identification of photovoltaic models. In [25], self-adaptive Jaya was implemented in design optimization for thermal devices. In [26], Jaya was guided for maximum power point tracking for PV system. In [27], Jaya was applied to optimize the coefficients of proportional plus integral controller and filter parameters of photovoltaic fed distributed static compensator.

In this paper, an improved Jaya (IJaya) algorithm is proposed and used to solve the ELD problem considering valve-point effect and multi-fuel options. There are three modifications in IJaya. The first one is, we introduce the concept of distance-varying acceleration coefficient (DVAC), which is determined by the best position and worst position in Jaya. The major purpose of this modification is to adjust the population’s trends of running towards the best position and escaping from the worst position. This coefficient assists the population with accelerating the convergence rate in the early stage and implementing enhanced local search in the later stage. The second one is, we introduce a self-adaptive population mechanism (SAPM) to determine the appropriate value of population size \( pop\_size \) automatically without manual tuning. Hence, researchers do not have to waste time on selecting values for \( pop\_size \). With SAPM, \( pop\_size \) is either increased or decreased according to the positive or negative value of \( d_w \). If \( pop\_size \) is increased, it means a certain number of populations need to be generated. Here we adopt an opposition-based learning (OBL) to generate the new population. If \( pop\_size \) is decreased, it means a certain number of population need to be deleted. Here we only keep the population with better fitness values to move to the next generation, and delete the worse ones. The third one is, we introduced Gaussian and Cauchy mutation (GCM) with distance-varying characteristic to improve the quality of the best solution in each iteration. With mutation operators, the best solution is able to jump out of local optima hence prevent premature problem. Specifically, the main contributions of this paper are as follows.

- IJaya algorithm is proposed by cooperating with DVAC, SAPM and GCM operators.
- IJaya is used to solve the ELDs considering valve-point effect and multi-fuel options.
- The superiority of IJaya to Jaya is proved by the comparisons of the results for ELDs.

Rest of this study is arranged as follows. In Section 2, the formulation of ELD problem is illustrated. Related works on Jaya and IJaya are showed in Section 3 and 4 respectively. Testing results and comparisons for ELDs are presented in Section 5. Conclusions are given in Section 6.

II. PROBLEM FORMULATION

The economic load dispatch (ELD) problem is expressed as an objective function to minimize the total fuel cost while satisfying different constraints as power balance constraint, generating capacity limit, ramp rate limit, valve-point effect and prohibited operating zones (POZs).

A. OBJECTIVE FUNCTION

Mathematical model for the ELD problem is to add up all the fuel costs of the generators in power system as expressed below [28]:

\[
\min \quad O = \sum_{i=1}^{N} O_i(X_i) \quad (1)
\]

where \( N \) is the total number of committed generators, \( i \) is the index of generator where \( i \in \{1, 2, ..., N\} \), \( X_i \) is the power output of generator \( i \), \( O_i(X_i) \) is the cost function of generator \( i \) with power output \( X_i \), \( O \) is the total cost of all the generators.

Generally speaking, in classical ELD, the cost function of each generator is described by quadratic polynomial as:

\[
O_l(X_i) = a_i X_i^2 + b_i X_i + c_i \quad (2)
\]

where \( a_i, b_i, c_i \) is the fuel cost coefficients of generator \( i \).

In practice, the valve-point effects on the costs of generators must be considered. So the rectified sinusoidal component is added to the classical cost function as follows:

\[
O_l(X_i) = a_i X_i^2 + b_i X_i + c_i + |e_i \times \sin(f_i \times (X_i^{\text{min}} - X_i))| \quad (3)
\]

where \( e_i, f_i \) are the fuel cost coefficients of generator \( i \) reflecting valve-point effect.

In some cases, the committed generators may be supplied by multiple fuels as natural gas, coal or oil. Then the cost function is defined with piecewise quadratic functions which reflect the effects of the fuel type changes [2]. Considering valve-point effect and multiple fuels, the objective function can be described as:

\[
O_l(X_i) = \begin{cases} 
O_{l_1}(X_i), & \text{fuel 1, } X_i^{\text{min}} \leq X_i \leq X_i^{1,1} \\
O_{l_2}(X_i), & \text{fuel 2, } X_i^{1,1} \leq X_i \leq X_i^{1,2} \\
\vdots \\
O_{l_m}(X_i), & \text{fuel m, } X_i^{m-1} \leq X_i \leq X_i^{m} 
\end{cases} \quad (4)
\]

where \( m \) is the total number of fuel types. \( O_{l_m}(X_i) \) as is the same as in Eq.(2) without considering valve-point effects, or as in Eq.(3) with considering valve-point effects.

B. CONSTRAINED FUNCTIONS

1) POWER BALANCE

The total power generated by the committed generators must equal to the summation of the demanded power \( X_{\text{demand}} \)
and the total transmission power loss $X_{\text{loss}}$, which can be formulated as:

$$
\sum_{i=1}^{N} X_i = X_{\text{demand}} + X_{\text{loss}}
$$

(5)

where $X_{\text{loss}}$ is calculated by Kron’s formula:

$$
X_{\text{loss}} = \sum_{i=1}^{N} \sum_{j=1}^{N} B_{ij} X_i + \sum_{i=1}^{N} B_{0i} X_i + B_{00}
$$

(6)

where $B_{ij}$, $B_{0i}$, $B_{00}$ are the B-matrix coefficients for $X_{\text{loss}}$, which can be generally assumed to be constants under a normal operating condition.

2) GENERATING CAPACITY

The power output $X_i$ should be within the generator’s maximum and minimum limits, as shown below:

$$
X_i^{\text{min}} \leq X_i \leq X_i^{\text{max}}
$$

(7)

where $X_i^{\text{max}}$ and $X_i^{\text{min}}$ are the maximum and minimum limits of the $i^{th}$ generator.

3) RAMP RATE LIMIT

Under practical circumstance, the operating range of every generator is restricted by its ramp rate limit, so the output power $X_i$ can not be adjusted instantaneously. The up-ramp and down-ramp constraints are as follows:

$$
X_i - X_i^0 \leq UR_i \quad \text{and} \quad X_i^0 - X_i \leq DR_i
$$

(8)

where $X_i$ is the present power output, $X_i^0$ is the previous power output, $UR_i$ and $DR_i$ is the up-ramp and down-ramp limit of generator $i$ respectively.

4) PROHIBITED OPERATING ZONES (POZs)

In practice, since there are physical limitations when operating the generator, the whole operating zones are not always available. Prohibited operating zones (POZs) lead to discontinuous regions for the objective function. The output power $X_i$ has constraints as follows:

$$
X_i \in \left\{ X_i^{\text{min}} \leq X_i \leq X_i^{\text{lower}}, X_i^{\text{upper}} \leq X_i \leq X_i^{\text{lower}}, X_i^{\text{lower}} \leq X_i \leq X_i^{\text{max}} \right\}
$$

(9)

where $z_i$ is the total number of POZs for generator $i$, $r$ is the index of POZs where $r \in [1, z_i]$, $X_i^{\text{lower}}$ and $X_i^{\text{upper}}$ are the lower and upper bounds of the $r^{th}$ POZ of the $i^{th}$ generator respectively.

III. JAYA ALGORITHM

The working principle of Jaya is explained as follows [18]. Suppose the fitness function $O(J)$ is required to be minimized or maximized. Let the design variable number is $var\_num$ where the index $u \in \{1, 2, ..., var\_num\}$, let the population size is $pop\_size$ where the index $v \in \{1, 2, ..., pop\_size\}$, let the maximum iteration number is $max\_iter$ where the index $w \in \{1, 2, ..., max\_iter\}$. Then let $J_{u,v,w}$ be the value of the $\nu^{th}$ variable for the $v^{th}$ candidate population during the $w^{th}$ iteration, then the new modified value $J_{u,v,w}$ is calculated by:

$$
J_{u,v,w} = J_{u,v,w} + r_1 \times (J_{u,\text{best},w} - |J_{u,v,w}|) - r_2 \times (J_{u,\text{worst},w} - |J_{u,v,w}|)
$$

(10)

where $J_{u,\text{best},w}$ is the updated value of $J_{u,v,w}$. $r_1$ and $r_2$ are two uniformly generated random numbers ranged in $[0, 1]$. $J_{u,\text{best},w}$ is the best solution with the best fitness value $O(J_{u,\text{best},w})$ and $J_{u,\text{worst},w}$ is the worst solution with the worst fitness value $O(J_{u,\text{worst},w})$.

It should be explained that, in Eq.(10), the first term $"J_{u,v,w}"$ represents the original position, which provides the necessary start point for each population (each population can be seen as a moving particle) to roam among the fitness space. The second term $"r_1 \times (J_{u,\text{best},w} - |J_{u,v,w}|)"$ encourages the population to fly toward the spot of the best position found so far. The third term $"-r_2 \times (J_{u,\text{worst},w} - |J_{u,v,w}|)"$ represents the tendency of the population to escape from the worst position found so far.

Usually, the initial populations are generated with random values, then the fitness value of each population is evaluated according to a pre-defined fitness function. Then for the next iteration, all the populations are updated according to Eq.(10). If the updated population obtains a better fitness value than its original fitness value, then the updated population is maintained and goes to the next iteration. Otherwise, keep the original population [18]. Pseudo code is shown in Algorithm 1.

**Algorithm 1 Jaya**

Initialize $var\_num$, $pop\_size$ and $max\_iter$;
Generate initial population $J$;
Evaluate the fitness value $O(J)$;
Set $w = 1$;
while $w < max\_iter$ do
 Identify $J_{u,\text{best},w}$ and $J_{u,\text{worst},w}$ according to $O(J)$;
 for $v = 1 \rightarrow pop\_size$ do
  for $u = 1 \rightarrow var\_num$ do
   Generate updated solution $J_{u,v,w}^*$ by Eq.(10);
  end
  if $O(J_{u,v,w}^*)$ is better than $O(J_{u,v,w})$ then
   $J_{u,v,w} = J_{u,v,w}^*$
   $O(J_{u,v,w}) = O(J_{u,v,w}^*)$
  else
   Keep the old value;
  end
 end
 $w = w + 1$;
end

IV. IMPROVED JAYA ALGORITHM (IJaya)

In this section, the authors introduced distance-varying acceleration coefficient (DVAC), self-adaptive population mechanism (SAPM), opposition-based learning (OBL), Gaussian
and Cauchy mutation (GCM). They were applied to standard Jaya to form the improved Jaya (Jaya) algorithm. In the following, the major components of Jaya are described in details respectively.

### A. DISTANCE-VARYING ACCELERATION COEFFICIENT (DVAC)

We can observe from Eq.(10) that, searching process towards better positions by Jaya is mainly guided by two stochastic terms, one is the best position \( J_{u,best,w} \) and the other one is the worst position \( J_{u,worst,w} \). Therefore, reasonable control of these two terms is crucial in searching for optimum solution efficiently and accurately.

Generally speaking, during the searching process, each population is expected to approach the promising region at the early stage. At the latter stage, since the population has converged to the promising area, fine-tuning should be implemented aiming to find the global optima. In order to meet the requirements, a distance-varying acceleration coefficient (DVAC), denoted as \( d_w \), is introduced to adjust the degree of approaching the best position and avoiding the worst position. The mathematical representation of \( d_w \) is given by:

\[
d_w = \begin{cases} \left( \frac{O(J_{u,best,w})}{O(J_{u,worst,w})} \right)^2, & \text{if } O(J_{u,worst,w}) \neq 0 \\ 1, & \text{otherwise} \end{cases}
\]

(11)

where \( O(J_{u,best,w}) \) and \( O(J_{u,worst,w}) \) are the fitness values of the best solution and worst solution in \( w \)th iteration by Jaya. Then we use Eq.(12) to replace Eq.(10):

\[
J'_{u,v,w} = J_{u,v,w} + r_1 \times (J_{u,best,w} - |J_{u,v,w}|) - d_w \times r_2 \times (J_{u,worst,w} - |J_{u,v,w}|)
\]

(12)

We can tell that, \( d_w \) has self-adaptive feature and its value increases gradually, since the distance between the best position and the worst position is becoming closer as the search process. Therefore, when \( d_w \) is small at the early stage, a relatively small term of the worst position, compared with the best position, will result in significantly accelerated speed in approaching the best position. In contrast, when \( d_w \) is gradually increasing to 1, it will fairly make the balance between the best position and the worst position, so the population would make use of both of the two sides to refine its current position at the latter stage. In addition, since the value of \( d_w \) is calculated adaptively, thus no additional parameter is introduced [24].

With the introduction of \( d_w \) to Jaya, significant improvements on the convergence rate and the optimum value have been observed, which is presented and discussed in Section 5.

### B. SELF-ADAPTIVE POPULATION MECHANISM (SAPM)

In Jaya, the determination of an appropriate value for \( pop_{size} \) is quite a difficult work, because it is related to how complex the problem is, which is hard to know in advance. In order to make the determination of \( pop_{size} \) easier, a self-adaptive Jaya algorithm was proposed in [25].

The most important feature of self-adaptive Jaya is the mechanism which determines the value of \( pop_{size} \) automatically without manual tuning. In this paper, the authors also applied self-adaptive population mechanism (SAPM) to Jaya, with purpose to reduce the number of parameters to be adjusted. However, different from [25], this paper has introduced opposition-based learning (OBL) to generate new populations when the value of \( pop_{size} \) is increased.

The new population size is denoted as \( pop_{size,new} \), which is calculated as follows:

\[
pop_{size,new} = \text{round}(pop_{size} + \Delta q \times pop_{size})
\]

(13)

where \( pop_{size} \) is the current population size, \( \Delta q \) is a random value between \([-0.5, 0.5]\) acting as a comparative development rate for population size.

Compared with the \( pop_{size} \) which is a constant value, \( pop_{size,new} \) may increase or decrease according to the positive or negative value of \( \Delta q \). When \( \Delta q \) is positive \( (pop_{size,new} > pop_{size}) \), all of the existing population will go into the next generation, and OBL is implemented to generate the remaining \( (pop_{size,new} - pop_{size}) \) solutions. When \( \Delta q \) is negative \( (pop_{size,new} < pop_{size}) \), only the population with better fitness values will go to the next generation. When \( \Delta q \) equals to zero \( (pop_{size,new} = pop_{size}) \), no change takes place.

It should be noted that, if \( pop_{size,new} \) keeps on decreasing and becomes smaller than the design variable number \( var_{num} \), then \( pop_{size,new} \) is reset to be equal to \( var_{num} \) (if \( pop_{size,new} < var_{num} \), then \( pop_{size,new} = var_{num} \)), in case of being stuck at local optima. Pseudo code of SAPM is shown in Algorithm 2. The way of generating new population by OBL will be discussed in the next section.

#### Algorithm 2 Update_Population (pop_size)

\[
\Delta q = \text{rand}([-0.5, 0.5])
\]

Calculate \( pop_{size,new} \) according to Eq.(13):

- if \( \Delta q > 0 \) then
  - Opposite_Learning \( (pop_{size,new}) \);
- else if \( \Delta q < 0 \) then
  - Order \( J \) by its fitness value;
  - Remove \( (pop_{size,new} - pop_{size}) \) population with worse fitness values;
  - \( pop_{size} = pop_{size,new} \);
- else
  - keep \( J \);
end

### C. OPPOSITION-BASED LEARNING (OBL)

Opposition-based learning (OBL) is usually utilized by population-based algorithm by calculating and evaluating the current population and its opposite population simultaneously, and choose the better one for going to next generation. It has successfully obtained better results in
biogeography-based optimization (BBO) [4], whale optimization algorithm (VOA) [29] and Krill herd algorithm (KH) [30], [31]. Here is the working principle. Suppose \( X = (X_1, X_2, \ldots, X_u) \) is a point with \( u \) variables in \( u \)-dimensional search space, where \( X_m \in [A_m, B_m] \) and \( m = 1, 2, \ldots, u \). Then the opposite point of \( X \) is represented as \( X' = (X'_1, X'_2, \ldots, X'_u) \), which is calculated by:

\[
X'_m = A_m + B_m - X_m \tag{14}
\]

In this paper, Eq.(14) is applied to the current population \( J \) to generate the opposite population \( J' \). To illustrate it in details, we suppose the current population \( J_{u,v,w} = (J_{1,v,w}, J_{2,v,w}, \ldots, J_{u,v,w}) \), the corresponding opposite solution can be defined as \( J'_{u,v,w} = (J'_{1,v,w}, J'_{2,v,w}, \ldots, J'_{u,v,w}) \), which is obtained by the following equation:

\[
J'_{u,v,w} = s \times (A_{u,v,w} + B_{u,v,w}) - J_{u,v,w} \tag{15}
\]

where \( J_{u,v,w} \) is the \( u \)th variable of the \( v \)th population in the \( w \)th iteration by Jaya. \( J'_{u,v,w} \) is the new generated opposite population, \( s \) is a random number in \([0, 1]\). \( A_{u,v,w} \) and \( B_{u,v,w} \) are the dynamic bounds of \( u \)th variable in the \( w \)th iteration for all the population, which can be obtained by the following equations:

\[
A_{u,v,w} = \min(J_{u,v,w}), \quad B_{u,v,w} = \max(J_{u,v,w}) \tag{16}
\]

By OBL, the searching space is expanded and the diversity of the population is strengthened. In this work, OBL is used in two aspects. The first one, when we are generating initial population for Jaya, we apply OBL simultaneously to get its opposite population. Then by comparing the current population with its opposite population, we choose the better one as the initial population. Pseudo code of the opposite initialization is shown in Algorithm 3. The second one, OBL is applied to the current population during the whole iteration process, with the aim of jumping to a new position which may have greater opportunity to get closer to the optimal solution. By comparing its fitness value, the fittest \( \text{pop}_\text{size}_\text{new} \) solutions are saved to the next iteration and the others are removed. Pseudo code of the opposite learning strategy is shown in Algorithm 4.

**D. GAUSSIAN AND CAUCHY MUTATION (GCM)**

It has been observed from simulations that, Jaya algorithm is able to quickly converge to a good spot, but the population sometimes keeps on staying at the same spot for a certain number of iterations without any improvement. To avoid this phenomenon, we introduce a mutation operator to provide additional diversity to help the population jump out of the stuck spot. Further, to address the difficulties of selecting appropriate step size for mutation for different problems, the step size of mutation in this paper has a distance-varying characteristic, so it is called distance-varying mutation (DVM). When the current best solution is not improving with increasing number of iterations, DVM is added to the current best solution to provide additional diversity and to get rid of the stuck spot.

Actually, different forms of mutation were used in different algorithms. In [30], Cauchy mutation was used with krill herd (KH) to prevent premature problem. In [32], Gaussian mutation was adopted into PSO (PSO-GM) to avoid local optima. In this paper, Gaussian and Cauchy mutation (GCM) are combined together with distance-varying characteristic, and it is implemented in Jaya algorithm to further broaden Jaya’s population diversity as well as to improve Jaya’s searching ability.

The probability density function of Gaussian distribution (Normal distribution) is described as follows:

\[
f(x) = \frac{1}{\sqrt{2\pi} \sigma^2} \exp[-(x-u)^2/(2\sigma^2)] \tag{17}
\]

where \( u \) is the mean value, \( \sigma^2 \) is the variance. \( N(u, \sigma^2) \) represents the normal distribution.

The Cauchy distribution probability density function is defined as follows:

\[
f(x) = \frac{\gamma}{\pi \gamma^2 + (x-x_0)^2} \tag{18}
\]
where $\gamma > 0$ is a proportion parameter, $x_0$ represents the position of peak. $C(x_0, \gamma^2)$ represents the Cauchy distribution.

By comparing the distribution of $G(0, 1)$ and $C(0, 1)$ from Fig.1, it’s obvious that $C(0, 1)$ has a wider variation scale than $G(0, 1)$ due to its long flat tails. Hence, Cauchy mutation is expected to perform a wider range search in the early stage, which can effectively make up the loss of population diversity and help the population escape from local optimum or moving away from a stuck-plateau. On the other hand, the smaller peak around the center shows that Cauchy mutation spends less time in exploiting the neighborhood and thus the fine-tuning ability is weaker than Gaussian mutation in small regions. As a result, Gaussian mutation is dominated for comprehensive local exploitation at the later stage [33]. The mathematical representation of GCM is given by:

$$
J'_{u,v,w} = J_{u,v,w} \times (1 + \theta \times d_w \times G(0, 1) + (1 - d_w) \times C(0, 1)) \quad (19)
$$

where $G(0, 1)$ is the random number that obeys standard Gaussian distribution, $C(0, 1)$ is the random number that obeys standard Cauchy distribution, $\theta$ is an adjusting coefficient which is set to be 0.01, and the distance-varying parameter $d_w$ is given by Eq.(11).

Algorithm 5 Get_Mutation ($J_{u,v,w}$)

| 1 | $m_\text{counter} = 0$ ; |
|---|---|
| 2 | Mutate $J_{u,v,w}$ according to Eq.(19) ; |
| 3 | Calculate the fitness value $O(J'_{u,v,w})$ ; |
| 4 | if $O(J'_{u,v,w})$ is better than $O(J_{u,v,w})$ then |
| 5 | $J_{u,v,w} = J'_{u,v,w}$ ; |
| 6 | $O(J_{u,v,w}) = O(J'_{u,v,w})$ ; |
| 7 | $m_\text{counter} = m_\text{counter} + 1$ ; |
| 8 | else |
| 9 | Keep the old value ; |
| 10 | end |

Pseudo code of GCM is shown in Algorithm 5. It is to be noted that, $m_\text{counter}$ is used to record the total number of replacements by GCM within one time of independent run, which will be discussed in details in Section 6.2.

Algorithm 6 IJaya

| 1 | Initialize var_num, pop_size and max_iter ; |
| 2 | Opposite_Initializing (var_num,pop_size) ; |
| 3 | Evaluate the fitness value $O(J)$ ; |
| 4 | Identify $J_{u,\text{best},w}$ and $J_{u,\text{worst},w}$ according to $O(J)$ ; |
| 5 | Set $w = 1$ ; |
| 6 | while $w < \text{max}_\text{iter}$ do |
| 7 | for $v = 1 \rightarrow \text{pop}_\text{size}$ do |
| 8 | for $u = 1 \rightarrow \text{var}_\text{num}$ do |
| 9 | Calculate $d_w$ by Eq.(11) ; |
| 10 | Generate updated solution $J'_{u,v,w}$ by Eq.(12) ; |
| 11 | end |
| 12 | if $O(J'_{v,w})$ is better than $O(J_{v,w})$ then |
| 13 | $J_{v,w} = J'_{v,w}$ ; |
| 14 | $O(J_{v,w}) = O(J'_{v,w})$ ; |
| 15 | else |
| 16 | Keep the old value ; |
| 17 | end |
| 18 | Identify $J_{u,\text{best},w}$ and $J_{u,\text{worst},w}$ according to $O(J)$ ; |
| 19 | Get_Mutation ($J_{u,\text{best},w}$) ; |
| 20 | Update_population (pop_size) ; |
| 21 | $w = w + 1$ ; |

E. IMPLEMENTATION OF IJAYA FOR ELD PROBLEM

According to the previous work, an improved Jaya (IJaya) algorithm is proposed. Algorithm 6 shows the pseudo code of IJaya. It starts by setting values for common parameters. Then the initial population is created and evaluated. Then we add DVAC to modify the Jaya function. After that, each population utilises the modified Jaya function to update its current value. Then we apply GCM to the best solution among the population. Next we use SAPM to update the value of pop_size for next iteration. If pop_size is increased, OBL will be used to generate a certain number of new population. At last, if the maximum iteration number max_iter is reached, stop the iteration and record the best solution. Otherwise, re-calculate $d_w$ and go to the next iteration. Fig.2 shows the flowchart of the procedures, which are illustrated with further details below:

Step 1: Set parameters. Three common parameters are initialized, they are number of design variable var_num, population size pop_size and maximum iteration number max_iter.

Step 2: Initialization. Initial population $J$ is generated in the form by:

$$
J_{u,v,w} = x_{u}^{\text{min}} + (x_{u}^{\text{max}} - x_{u}^{\text{min}}) \times \text{rand}(\text{pop}_\text{size}, \text{var}_\text{num}) \quad (20)
$$
where $J_{u,v,w}$ is the $u^{th}$ generator in the $v^{th}$ solution, $X_{u}^{min}$ and $X_{u}^{max}$ are the lower and upper limits of the $u^{th}$ generator given by the constraints of generating capacity limits.

Step 3: Apply OBL. Update $J$ according to Algorithm 3.

Step 4: Evaluation. Fitness value of $J$ is calculated by objective function, which is problem dependent. In Section 5, we use Eq.(3) as the objective function for Case I and Case II, we use Eq.(2) as the objective function for Case III and Case IV, we use Eq.(4) as the objective function for Case V.

Step 5: Identify $J_{u,\text{best},w}$ and $J_{u,\text{worst},w}$ within $J$.

Step 6: Apply DVAC. Calculate $d_w$ by Eq.(11).

The current population $J$ is updated by Eq.(12).

Step 7: Comparison. Calculate the fitness value of each population, compare the present fitness value with its former fitness value, if the fitness value gets better, keep the present population; otherwise, keep the former population.

Step 8: Identify $J_{u,\text{best},w}$ and $J_{u,\text{worst},w}$ within $J$.

Step 9: Apply GCM. Update the current best population $J_{u,\text{best},w}$ according to Algorithm 5.

Step 10: Update $J$. Update the current population $J$ by SAPM according to Algorithm 2.

Step 11: Check the stopping condition. If the max_iter is reached, stop the loop and report the best solution; otherwise set $w = w + 1$ and go to Step 5 for re-calculate $d_w$.

V. NUMERICAL EXPERIMENTS

Since the proposed IJaya algorithm is the hybridization of Jaya, DVAC, SAPM and GCM, it is quite necessary to observe the relative effectiveness of each constituent, hence five different algorithms are experimented respectively.

- Jaya: The standard Jaya algorithm.
- D-Jaya: Jaya with DVAC.
- S-Jaya: Jaya with SAPM.
- GC-Jaya: Jaya with GCM and DVAC.
- IJaya: Jaya with DVAC, SAPM and GCM.
All the cases are developed using MATLAB software (version 2017a) and executed on a computer under Windows 7 on Intel(R) Core(TM) i5-7500 CPU with 8GB RAM environment.

A. CASE 1
Case 1 consists of a 13-generators system with load demand of 2520MW. The independent run time run_time is 100, var_num is 13, pop_size is 50, max_iter is 5000. System data is shown in [34].

Fig.3 shows the convergence feature of $F_{min}$ by Jaya and its variants. We can observe that, IJaya performs the fastest convergence rate, followed by GC-Jaya and S-Jaya, who achieved almost the same minimum value in the end but they both require more times of iterations. D-Jaya reaches the similar minimum value, but it is much slower.

Fig.4 and Fig.5 provide the value distributions of $F_{min}$ over 100 running times by each algorithm. We can observe that, the best robustness is made by IJaya, since most of the individuals are closest to the best value and the standard deviation is the smallest. The worst performer is Jaya, who is far from the best value and varies much more than the others.

Table 1 shows the outputs for $F_{min}$ by Jaya and its variants. Table 2 presents the comparisons with other algorithms in $F_{min}, F_{ave}, F_{max}, Std$ and $T$. The best results are shown in bold. We can observe that, IJaya is successful in obtaining the best $F_{min}$, but fails in $F_{ave}, F_{max}$ and $Std$.

Actually, FAMPSO [39] surpasses IJaya in $F_{ave}$ and $F_{max}$, while DSPSO-TSA [41] exceeds IJaya in $Std$. The reason is that, GCM operator can help IJaya jump out of local optima when premature problem happens, hence IJaya owns strong capability in keeping on reaching the global optima than the compared algorithms. However, IJaya lacks abilities of maintaining good robustness, so the amplitude change of the
final results of IJaya is relatively large, resulting worse $F_{ave}$, $F_{max}$ and $Std$.

**B. CASE 2**

Case 2 is a 40-generators system with 10500MW load demand. The independent run time $run\_time$ is 100, $var\_num$ is 40, $pop\_size$ is 100, $max\_iter$ is 5000. The related system data is provided in [34].

![FIGURE 6. Convergence behaviors for Case 2 by Jaya and its variants.](image)

From Fig.6 we can see, the five algorithms are all competitive in convergence rate. But IJaya shows the strongest ability in reducing the cost and outperforms the others in solution quality.

**TABLE 3. Outputs for $F_{min}$ in Case 2 by Jaya and its variants.**

| Unit  | Jaya | D-Jaya | S-Jaya | GC-Jaya | IJaya |
|-------|------|--------|--------|---------|-------|
| 1     | 111.433 | 112.8810 | 113.9996 | 112.6555 | 113.0249 |
| 2     | 111.8901 | 111.1874 | 113.9999 | 113.7768 | 111.4273 |
| 3     | 96.6984 | 97.4361 | 119.9998 | 97.8727 | 97.6397 |
| 4     | 180.1071 | 179.8098 | 179.9333 | 179.8188 | 179.9077 |
| 5     | 90.0475 | 91.9587 | 97.0000 | 96.8854 | 89.3514 |
| 6     | 139.5192 | 139.9943 | 140.0000 | 139.9193 | 139.9996 |
| 7     | 296.7450 | 299.9850 | 259.6129 | 299.9456 | 259.9181 |
| 8     | 285.6460 | 284.6554 | 286.1142 | 285.1865 | 284.8410 |
| 9     | 286.3665 | 284.7994 | 284.6013 | 285.8114 | 284.7114 |
| 10    | 130.7690 | 130.0188 | 130.0093 | 130.0777 | 130.0162 |
| 11    | 94.1220 | 94.0022 | 94.0001 | 94.1005 | 168.9855 |
| 12    | 94.2387 | 94.0130 | 94.0001 | 94.0500 | 94.0011 |
| 13    | 125.2388 | 125.0057 | 125.0001 | 125.0232 | 125.0233 |
| 14    | 304.8618 | 394.2809 | 394.2793 | 394.3210 | 394.4646 |
| 15    | 393.9342 | 304.5297 | 394.2794 | 394.3440 | 394.2457 |
| 16    | 483.2159 | 484.0512 | 394.2796 | 394.5101 | 394.3681 |
| 17    | 490.8335 | 490.3617 | 489.2802 | 489.3216 | 489.3442 |
| 18    | 485.1780 | 489.2845 | 489.2794 | 489.5964 | 489.2860 |
| 19    | 511.8705 | 511.2839 | 511.2797 | 511.3489 | 511.2627 |
| 20    | 511.3752 | 511.2904 | 511.2600 | 511.2318 | 511.4584 |
| 21    | 524.0100 | 523.2840 | 523.2601 | 523.8667 | 523.4239 |
| 22    | 523.8054 | 523.3001 | 523.2803 | 523.3245 | 523.7759 |
| 23    | 523.3344 | 523.2825 | 523.2600 | 523.2738 | 523.7398 |
| 24    | 523.5209 | 523.3000 | 523.2601 | 523.5297 | 523.3627 |
| 25    | 522.7360 | 523.3365 | 523.2795 | 523.4524 | 523.2396 |
| 26    | 523.5990 | 523.2956 | 523.2803 | 523.5207 | 523.3280 |
| 27    | 10.3253 | 10.0006 | 10.0000 | 10.1476 | 10.0211 |
| 28    | 10.2294 | 10.0011 | 10.0001 | 10.0737 | 10.0000 |
| 29    | 10.8754 | 10.0101 | 10.0000 | 10.0449 | 10.0000 |
| 30    | 90.6468 | 89.1445 | 97.0000 | 96.4813 | 89.8714 |
| 31    | 189.8277 | 189.9938 | 190.0000 | 189.9938 | 189.9291 |
| 32    | 189.0318 | 189.9949 | 190.0000 | 189.9835 | 189.9907 |
| 33    | 189.6082 | 189.9981 | 190.0000 | 189.9279 | 189.9930 |
| 34    | 199.5223 | 199.9990 | 199.9997 | 199.7655 | 164.6567 |
| 35    | 197.6694 | 199.9779 | 200.0000 | 190.2947 | 199.9986 |
| 36    | 199.4316 | 199.9627 | 200.0000 | 199.8252 | 199.9586 |
| 37    | 109.5700 | 109.9933 | 110.0000 | 109.9756 | 109.9954 |
| 38    | 109.9437 | 109.9969 | 110.0000 | 109.9259 | 109.9432 |
| 39    | 109.7378 | 109.9771 | 109.9999 | 109.9459 | 109.9979 |
| 40    | 512.3499 | 511.3139 | 511.2801 | 511.4744 | 511.3086 |
| 41    | 121715.4003 | 121596.4485 | 121571.6513 | 121568.8609 | 121434.3785 |

![FIGURE 7. Trails of 100 runs for Case 2 by Jaya and its variants.](image)

From Fig.7 and Fig.8 we can see, the robustness and coherence by all the five algorithms are considerably acceptable. However, IJaya behaves the best because all its individuals are kept closest at the best value. In D-Jaya, there are two outliers with extreme values plotted as “+” symbol, which means D-Jaya suffers problems of falling into local optima with very bad values.

Table.3 shows the outputs for $F_{min}$. Table.4 shows the comparison with state-of-the-art methods. In this case, the best value of $F_{min}$, $F_{ave}$ and $F_{max}$ are all achieved by IJaya, mainly because of the broadened population diversity by SAPM and the accelerated convergence rate by DVAC, which greatly ensures the solution quality. However, when it comes to $Std$, IJaya is less competitive than CTLBO [9] and ranks the second place as 173.7012, it means that the performance of IJaya still has space for improvement in this case.

**C. CASE 3**

Case 3 consists of a 6-generators system with load demand of 1263MW. Transmission loss and prohibited operating zones (POZs) are considered in this case, so the number of non-convex decision spaces is increased, causing more complexity in searching for global optima. The parameter of
TABLE 4. Comparison of fuel costs for Case 2 by different methods.

| Methods       | \(F_{\text{min}}\) | \(F_{\text{ave}}\) ($/h) | \(F_{\text{max}}\) | Std | \(T\) (s) |
|---------------|---------------------|----------------|-------------------|-----|--------|
| MPSO [3]      | 122252.2650         | NA             | NA                | NA  | NA     |
| TLBO [9]      | 124517.27           | 126581.56      | 128207.06         | 1060| NA     |
| CTILBO [9]    | 121535.83           | 121790.23      | 122116.18         | 150 | NA     |
| PSO [11]      | 122025.794          | 122558.455     | 123461.679        | NA  | 15.75  |
| NPSO [11]     | 121704.7391         | 122221.3697    | 122995.0976       | NA  | 3.82   |
| NPSO-LRS [11] | 121664.4308         | 122209.3185    | 122981.5913       | NA  | 3.93   |
| DEC-SQP [12]  | 121741.9793         | 122295.1278    | 122839.2941       | 386.18| 14.26 |
| MP-CJaya [21] | 121480.10           | 121861.08      | NA                | NA  | NA     |
| Jaya-SML [22] | 121476.3977         | 121695.0773    | 122039.8731       | 147.4901|12.89 |
| CPSO [32]     | 121855.11           | 122469.64      | 123767.56         | 307.15| NA     |
| PSO-GM [32]   | 121845.96           | 122398.36      | 123219.22         | 258.44| NA     |
| CBPSO-RVM [32]| 121553.32           | 122281.14      | 123094.98         | 259.99| NA     |
| EP-SQP [36]   | 122323.97           | 122379.63      | NA                | NA  | 997.73 |
| PPSO-SQP [36] | 122094.67           | 122245.25      | NA                | 783.97|        |
| SPSO [42]     | 122049.66           | NA             | NA                | NA  | NA     |
| PC-PSO [42]   | 121767.89           | NA             | NA                | NA  | NA     |
| SOH-PSO [42]  | 121301.14           | 121853.57      | NA                | 783.97|        |
| PSO [43]      | 121735.47           | 122513.92      | 123467.41         | NA  | NA     |
| APSO(1) [43]  | 121704.74           | 122221.37      | 122995.10         | NA  | NA     |
| APSO(2) [43]  | 121663.52           | 122153.67      | 122912.40         | NA  | NA     |
| HDE [44]      | 121813.26           | 122705.66      | NA                | 6.92 |        |
| ST-HDE [44]   | 121696.51           | 122204.30      | NA                | 6.07 |        |
| Jaya          | 121715.4003         | 122252.9462    | 123053.5788       | 304.95| 12.26 |
| D-Jaya        | 121596.4465         | 122202.7637    | 123136.8133       | 271.39| 7.19   |
| S-Jaya        | 121517.6513         | 121948.4265    | 122423.8331       | 193.57| 10.03  |
| GC-Jaya       | 121568.8609         | 121877.8997    | 122472.6624       | 210.54| 11.44  |
| IJaya         | 121454.3785         | 121770.3299    | 122109.0198       | 173.70| 14.04  |

run\_time is 100, var\_num is 6, pop\_size is 50, max\_iter is 5000. System data is taken from [45].

FIGURE 9. Convergence behaviors for Case 3 by Jaya and its variants.

From Fig. 9, it is clear that IJaya converges fastest, especially at the very early period within 100 iterations. Moreover, except for Jaya, all the others achieve competitive results at the end of the iteration.

From Fig.10 and Fig.11, we can see there are extreme values by Jaya and D-Jaya, which means their stability are comparatively worse. However, IJaya still makes the greatest improvements in maintaining the best results and the standard deviation is as small as nearly zero.

Table 5 shows that, compared with original Jaya and its variants, IJaya achieves the best schedule with minimum total generation \(P_{\text{total}}\) and minimum network loss \(P_{\text{loss}}\) in addition to minimum fuel cost \(F\). In Table.6 we can see that, among all the compared algorithms, IJaya obtains the best value of \(F_{\text{min}}\), \(F_{\text{ave}}\) and \(F_{\text{max}}\), except for Std, which is as similar as in Case 2. The reason has been presented in Case 2, so it is not repeated here.

D. CASE 4

In Case 4, 15-generators with 2630MW load demand is experimented. The constraints are as the same as in Case 3. The parameter of run\_time is 100, var\_num is 15, pop\_size is 100, max\_iter is 5000. System data is taken from [45].

In Fig.12, D-Jaya, S-Jaya and GC-Jaya are able to converge to the optimum value at a faster rate than Jaya, while IJaya achieves the fastest convergence rate and the least fuel cost within least number of iterations.

In Fig.13 and Fig.14, the value distribution and spacing of \(F_{\text{min}}\) are observed intuitively. It is clear that, IJaya not only achieves the best fuel cost but also maintains the best value

\[F_{\text{min}}, F_{\text{ave}}\quad \text{and} \quad F_{\text{max}}, \quad \text{except for Std, which is as similar as in Case 2. The reason has been presented in Case 2, so it is not repeated here.}\]
TABLE 6. Comparison of fuel costs for Case 3 by different methods.

| Methods        | $F_{\text{Total Cost}}$, $F_{\text{Std}}$, $F_{\text{Max}}$ | Std | $T$(s) |
|----------------|-------------------------------------------------|-----|------|
| ASS [5]        | 15448.00, 15472.00, 15459.70                    | 6.252 | NA   |
| PSO-LSR [11]   | 15450.00, 15454.00, 15455.00                     | NA  | NA   |
| NPSO-LSR [11]  | 15450.00, 15452.00, 15454.00                     | NA  | NA   |
| NPSO-LSR [11]  | 15540.00, 15542.50, 15545.00                     | NA  | NA   |
| MPS-Jaya [21]  | 15446.17, 15449.23, 15451.68                     | NA  | NA   |
| Jaya-SML [22]  | 15445.1662, 15447.2910, 15450.6497               | 6.221 | 2.11 |
| SPPO [42]      | 15466.63, 15532.64, 15642.68                     | NA  | 0.0062 |
| PC-PSSO [42]   | 15453.09, 15514.98, 15633.30                     | NA  | 0.0633 |
| SOH-PSSO [42]  | 15446.02, 15497.35, 15609.64                     | NA  | 0.0833 |
| SA [46]        | 15461.10, 15488.98, 15545.50                     | 28.637 | 50.36 |
| GA [46]        | 15457.96, 15477.71, 15524.69                     | 17.407 | 46.60 |
| TS [46]        | 15454.89, 15472.56, 15498.05                     | 13.7195 | 20.55 |
| PSO [46]       | 15450.14, 15463.83, 15491.71                     | 10.1502 | 6.82 |
| MTS [46]       | 15450.06, 15451.17, 15453.64                     | 0.9287 | 1.29 |
| PA [10]        | 15450.5090, 15452.5310, 15458.4427               | 2.048 | 1.965 |
| CMFA [10]      | 15449.8994, 15449.8994, 15459.8994               | 8.96E-2 | 2.724 |
| DE [47]        | 15449.7666, 15449.874, 15449.777                 | NA  | NA   |
| Jaya           | 15446.3387, 15457.6682, 15513.3994               | 13.29 | 2.29 |
| D-Jaya         | 15443.4793, 15456.3479, 15488.1453               | 11.77 | 1.14 |
| S-Jaya         | 15445.0080, 15448.3572, 15500.2037               | 8.32 | 2.66 |
| GC-Jaya        | 15443.3002, 15454.3031, 15468.7074               | 9.42 | 2.04 |
| IJaya          | 15443.0881, 15446.3730, 15448.4333               | 5.30 | 2.55 |

FIGURE 12. Convergence behaviors for Case 4 by Jaya and its variants.

FIGURE 13. Trails of 100 runs for Case 4 by Jaya and its variants.

Table 7 shows the outputs for $F_{\text{min}}$. Table 8 illustrates that the best value of $F_{\text{min}}$ and $F_{\text{ave}}$ and $F_{\text{Max}}$ are all achieved by IJaya. It is to be noted that, the best value of $F_{\text{Std}}$ is also achieved by IJaya, as small as 3.0849, which is significantly better than the other compared algorithms.

E. CASE 5

Case 5 consists of a 10-generators system with load demand of 2700MW. Different from aforementioned cases, the generators in this case have multi-fuel options, which is presented in [2]. The $\text{run-time}$ is 100, $\text{var_num}$ is 10, $\text{pop_size}$ is 50, $\text{max_iter}$ is 2000. System data is also taken from [2].

In Fig.15, all the algorithms gain acceptable convergence rate and show their superiority over Jaya, and once more, IJaya is the best performer.
FIGURE 15. Convergence behaviors for Case 5 by Jaya and its variants.

FIGURE 16. Trails of 100 runs for Case 5 by Jaya and its variants.

FIGURE 17. Boxplots of 100 runs for Case 5 by Jaya and its variants.

In Fig.16 and Fig.17, even though with multi-fuel options, D-Jaya, S-Jaya, GC-Jaya and IJaya are all superior to Jaya in terms of solution quality and robustness, while IJaya stays closest to the minimum $F$ and the Std reaches nearly zero.

Table 9 shows the optimal outputs. In Table 10, IJaya obtains the first place in ranking in terms of $F_{\text{min}}$, $F_{\text{ave}}$ and $F_{\text{max}}$. As for Std, IJaya ranks behind Jaya-SML [22]. But IJaya keeps the Std value as small as 0.0360, it means almost all the independent runs have converged to the best value, which actually outperforms Jaya and its variants greatly in this case.

VI. SOME DISCUSSIONS

A. AVERAGE NUMBER OF ITERATIONS

As we know, introducing new strategies will inevitably increase the computational complexity. Therefore, calculate

the average number of iterations is very important to evaluate the algorithm’s efficiency. In Case 1-5, the number of iteration for each time of independent run is recorded until the minimum fuel cost $F_{\text{min}}$ is achieved in the last iteration. And the average number of iterations by 100 independent runs is shown in Table 11.

| Methods     | $F_{\text{min}}$ | $F_{\text{ave}}$ | $F_{\text{max}}$ | Std | $T$ (s) |
|-------------|------------------|------------------|------------------|-----|---------|
| CGA-MU [2]  | 624.7193         | 627.6087         | 633.8652         | NA  | 26.64   |
| IGA-MU [2]  | 624.5176         | 625.8692         | 630.8705         | NA  | 7.32    |
| PSO-LRS [11]| 624.2297         | 625.7887         | 628.3214         | NA  | 0.88    |
| NPSO [11]   | 624.1624         | 625.2180         | 627.4237         | NA  | 0.35    |
| NPSO-LRS [11]| 624.1273        | 624.9985         | 626.9981         | NA  | 0.52    |
| Jaya-SML [22]| 623.9738        | 624.0466         | 624.1300         | 0.0327 | 7.31 |
| CPSO [32]   | 624.1715         | 624.5493         | 624.7844         | 0.1278 | NA    |
| P-SO-GM [33]| 624.3050         | 624.6749         | 625.0854         | 0.1580 | NA    |
| CPSO-RVM [32]| 623.9588        | 624.0816         | 624.2950         | 0.0376 | NA    |
| TSA [41]    | 624.3078         | 624.8285         | 635.0623         | 1.1593 | 9.71  |
| GA [41]     | 624.5050         | 624.7419         | 626.8169         | 0.1005 | 18.37 |
| PSO [41]    | 624.3045         | 624.5054         | 625.9325         | 0.1749 | 11.04 |
| PSO [43]    | 624.3506         | 625.8198         | 629.1073         | NA  | NA     |
| APSO(1) [43]| 624.1624         | 625.2180         | 627.4237         | NA  | NA     |
| APSO(2) [43]| 624.0145         | 624.8185         | 627.3097         | NA  | NA     |
| Jaya        | 624.6490         | 625.9327         | 637.4175         | 2.14 | 6.33   |
| D-Jaya      | 624.3476         | 624.8181         | 637.6690         | 2.01 | 2.22   |
| S-Jaya      | 624.4245         | 624.6912         | 628.2236         | 0.79 | 4.06   |
| GC-Jaya     | 624.2249         | 624.6812         | 628.2811         | 0.66 | 2.78   |
| IJaya       | 623.9383         | 624.0080         | 624.0646         | 0.0360 | 5.18  |

It is obvious to see that, in Table 11, IJaya achieves $F_{\text{min}}$ with the least number of iterations in all the cases. The average number of iterations by GC-Jaya is slightly more than S-Jaya in Case 2, but in the rest of the cases GC-Jaya is less than S-Jaya. Jaya shows the worst efficiency, because it spends almost the maximum number of iterations for each case. The comparisons prove that it is meaningful and
effective to introduce new strategies to help Jaya improve its iteration efficiency.

B. SUCCESS RATE OF GCM

To investigate how much improvement is provided by GCM, the average number of replacements over 100 running times is counted. The replacement only happens when the mutated population \( J_{u, best, w} \) provides better fitness value than \( J_{w, best, w} \). The success rate of GCM could be expressed as:

\[
\text{success_rate} = \frac{m\_\text{counter}}{m\_\text{total_num}}
\]

where \( success\_rate \) is the percentage of successful replacement, \( m\_\text{counter} \) is the average number of successful replacement, \( m\_\text{total_num} \) is the average number of iterations for one run (see Table.11).

| TABLE 12. Success rate of Gaussian and Cauchy mutation in IJaya. |
|---------------------|---------------------|---------------------|
| Case No. | \( m\_\text{counter} \) | \( m\_\text{total_num} \) | success_rate   |
|-------------|---------------------|---------------------|---------------------|
| Case 1      | 23                  | 3683                | 0.59%              |
| Case 2      | 52                  | 4737                | 1.10%              |
| Case 3      | 41                  | 3774                | 1.09%              |
| Case 4      | 107                 | 4027                | 2.66%              |
| Case 5      | 44                  | 1989                | 2.21%              |

From Table.12, it can be observed that the successful percentage in all the cases is relatively low. This is because in IJaya, only one time of mutation for \( J_{u, best, w} \) in each generation is not enough to produce offsprings which are superior to the father. However, in Case 4 and 5, the successful percentages are comparatively higher than the other cases. The reason is, Case 4 and Case 5 suffer bigger problems of falling into local minima because of more complex constrained conditions, so they would need more mutations to move the best particles to new positions to get away from the local minima. In fact, instead of only one particle (\( J_{u, best, w} \)) participating in GCM, letting more particles participate in the mutation operator may increase the success rate of GCM.

C. FRIEDMANN TEST

To specifically evaluate how much effectiveness does each algorithm have, a well-known mathematical tool called ‘Friedman test’ is employed to calculate the exact ranking of each algorithm. Here we adopt the average fuel cost \( F_{\text{ave}} \) to be tested. According to \( F_{\text{ave}} \) in Case 1-5, we can easily get the ranking in each case for each algorithm. Then we calculate the mean value as its final ranking for each algorithm.

The detailed rankings are provided in Table.13 and Fig.18. Conclusions can be made that the performance of IJaya is better than the others. It has obtained the first rank with average score of 1.00 in comparison to GC-Jaya (2.40), S-Jaya (2.60), D-Jaya (4.00) and Jaya (5.00).

FIGURE 18. Friedman test ranking of the average fuel cost for Case 1-5.

VII. CONCLUSION

This paper proposed an improved Jaya (IJaya) algorithm by introducing new strategies to the standard Jaya. To observe clearly the relative effectiveness of each strategy, D-Jaya, S-Jaya and GC-Jaya are also presented to be compared with IJaya. Based on the results of being applied on the ELD problem, we can conclude that, IJaya has significantly improved Jaya’s performance and provided better power outputs irrespective of the generator numbers and the constrained conditions.

Actually, there are three attractive properties of using IJaya to solve the ELD problems. Firstly, even though new strategies are added, there is no more parameter has been introduced throughout the whole implementation. Secondly, because every strategy is not complicated, the working principle of their combination with Jaya is easy to understand. Thirdly, the overall frame of IJaya can be easily transported to other population-based evolutionary algorithms (EAs). Since IJaya has gained great superiorities in solving the ELD problem, it is supposed to apply to other optimization problems, such as micro-grid power dispatch problems, global optimization problems of overcurrent relays and dust control systems, to further expand its applications. Particularly in the field of intelligent control of industrial dust in environmental protection, which is one of the authors’ interests in research in the future.

REFERENCES

[1] G. Abbas, J. Gu, U. Farooq, M. U. Asad, and M. El-Hawary, “Solution of an economic dispatch problem through particle swarm optimization: A detailed survey—Part I,” IEEE Access, vol. 5, pp. 15105–15141, 2017.
[2] C.-L. Chiang, “Improved genetic algorithm for power economic dispatch of units with valve-point effects and multiple fuels,” IEEE Trans. Power Syst., vol. 20, no. 4, pp. 1690–1699, Nov. 2005.
[3] J.-B. Park, K.-S. Lee, J.-R. Shin, and K. Y. Lee, “A particle swarm optimization for economic dispatch with nonsmooth cost functions,” IEEE Trans. Power Syst., vol. 20, no. 1, pp. 34–42, Feb. 2005.
[4] A. Bhattacharya and P. K. Chattopadhyay, “Biogeography-based optimization for different economic load dispatch problems,” IEEE Trans. Power Syst., vol. 25, no. 2, pp. 1064–1077, May 2010.
[5] B. K. Panigrahi, S. R. Yadav, S. Agrawal, and M. K. Tiwari, “A clonal algorithm to solve economic load dispatch,” Electr. Power Syst. Res., vol. 77, no. 10, pp. 1381–1389, Aug. 2007.

[6] X.-S. Yang, S. S. S. Hosseini, and A. H. Gandomi, “Firefly algorithm for solving non-convex economic dispatch problems with valve loading effect,” Appl. Soft Comput., vol. 12, no. 3, pp. 1180–1186, Mar. 2012.

[7] L. D. S. Coelho and V. C. Mariani, “An improved harmony search algorithm for power economic load dispatch,” Energy Convers. Manage., vol. 50, no. 10, pp. 2522–2526, Oct. 2009.

[8] D. C. Secui, “A new modified artificial bee colony algorithm for the economic dispatch problem,” Energy Convers. Manage., vol. 89, pp. 43–62, Jan. 2015.

[9] X. He, Y. Rao, and J. Huang, “A novel algorithm for economic load dispatch of power systems,” Neurocomputing, vol. 171, pp. 1454–1461, Jan. 2016.

[10] Y. Yang, B. Wei, H. Liu, Y. Zhang, J. Zhao, and E. Mania, “Chaos firefly algorithm with self-adaptation mechanism for solving large-scale economic dispatch with valve-point effects and multiple fuel options,” IEEE Access, vol. 6, pp. 45907–45922, 2018.

[11] A. L. Selvakumar and K. Thanushkodi, “A new particle swarm optimization solution to nonconvex economic dispatch problems,” IEEE Trans. Power Syst., vol. 22, no. 1, pp. 42–51, Feb. 2007.

[12] L. D. S. Coelho and V. C. Mariani, “Combining of chaotic differential evolution and quadratic programming for economic dispatch optimization with valve-point effect,” IEEE Trans. Power Syst., vol. 21, no. 2, pp. 989–996, May 2006.

[13] X. Shen, G. Wu, R. Wang, H. Chen, H. Li, and J. Shi, “A self-adapted across neighborhood search algorithm with variable reduction strategy for solving non-convex static and dynamic economic dispatch problems,” IEEE Access, vol. 6, pp. 41314–41324, Jul. 2018.

[14] M. Gholamghasemi, E. Akbari, M. B. Asadpoor, and M. Ghasemi, “A new solution to the non-convex economic load dispatch problems using phasor particle swarm optimization,” Appl. Soft Comput., vol. 79, pp. 111–124, Jun. 2019.

[15] F. Li, J. Qin, and Y. Kang, “Multi-agent system based distributed pattern search algorithm for non-convex economic load dispatch in smart grid,” IEEE Trans. Power Syst., vol. 34, no. 3, pp. 2093–2102, May 2019.

[16] F. Mohammadi and H. Abdi, “A modified crow search algorithm (MCSA) for solving economic load dispatch problem,” Appl. Soft Comput., vol. 71, pp. 51–65, Oct. 2018.

[17] G. Xiong and D. Shi, “Orthogonal learning competitive swarm optimizer for economic dispatch problems,” Appl. Soft Comput., vol. 66, pp. 134–148, May 2018.

[18] R. V. Rao, “Jaya: A simple and new optimization algorithm for solving constrained and unconstrained optimization problems,” Int. J. Ind. Eng. Comput., vol. 7, no. 1, pp. 19–34, 2016.

[19] J. Chen, J. H. Park, and S. Xu, “Stability analysis of systems with time-varying delay: A quadratic-partitioning method,” IET Control Theory Appl., vol. 13, no. 18, pp. 3184–3189, Dec. 2019.

[20] R. V. Rao, Jaya: An Advanced Optimization Algorithm and Its Engineering Applications. Cham: Springer, 2019.

[21] J. Yu, C.-H. Kim, A. Wadood, T. Khurshaid, and S.-B. Rhee, “A novel multi-population based chaotic JAYA algorithm with application in solving economic load dispatch problems,” Energies, vol. 11, no. 8, p. 1946, 2018.

[22] J.-T. Yu, C.-H. Kim, A. Wadood, T. Khurshaid, and S.-B. Rhee, “Jaya algorithm with self-adaptive multi-population and Levy flights for solving economic load dispatch problems,” IEEE Access, vol. 7, pp. 21372–21384, 2019.

[23] R. V. Rao and A. Saroj, “Multi-objective design optimization of heat exchangers using elitist-Jaya algorithm,” Energy Syst., vol. 9, no. 2, pp. 305–341, May 2018.

[24] K. Yu, J. J. Liang, B. Y. Qu, X. Chen, and H. Wang, “Parameters identification of photovoltaic models using an improved JAYA optimization algorithm,” Energy Convers. Manage., vol. 150, pp. 742–753, Oct. 2017.

[25] R. V. Rao and K. C. More, “Design optimization and analysis of selected thermal devices using self-adaptive Jaya algorithm,” Energy Convers. Manage., vol. 140, pp. 24–35, May 2017.

[26] C. Huang, L. Wang, R. S.-C. Yeung, Z. Zhang, H. S.-H. Chung, and A. Bensoussan, “A prediction model-guided Jaya algorithm for the PV system maximum power point tracking,” IEEE Trans. Sustain. Energy, vol. 9, no. 1, pp. 45–55, Jan. 2018.

[27] S. Mishra and P. K. Ray, “Power quality improvement using photovoltaic fed DSTATCOM based on JAYA optimization,” IEEE Trans. Sustain. Energy, vol. 7, no. 4, pp. 1672–1680, Oct. 2016.
DEXUAN ZOU received the B.S. degree in electronic information science and technology from Liaoning University, Shenyang, China, in 2005, and the M.S. degree in signal and information processing and the Ph.D. degree in control theory and control engineering from Northeastern University, Shenyang, China, in 2008 and 2011, respectively. He is currently an Associate Professor with the School of Electrical Engineering and Automation, Jiangsu Normal University, China. He has published over 50 journal articles, over 20 papers in conference proceedings, and one book. His current research interests include evolutionary algorithms and economic load dispatch.

CHUNLIANG LI is currently a Senior Engineer and EMBA working with Xuzhou Boyuan Technology Company Ltd. His current research interests include enterprise management and industrial dust control and optimization technology research.