Comparison between dynamic programming and genetic algorithm for hydro unit economic load dispatch

Bin XU1, Ping-an ZHONG2,3, Yun-fa ZHAO3, Yu-zuo ZHU4, Gao-qi ZHANG5

1. College of Hydrology and Water Resources, Hohai University, Nanjing 210098, P. R. China
2. National Engineering Research Center of Water Resources Efficient Utilization and Engineering Safety, Hohai University, Nanjing 210098, P. R. China
3. China Three Gorges Corporation, Beijing100038, P. R. China
4. Datang Yantan Hydropower Corporation, Nanning 530022, P. R. China
5. Yellow River Engineering Consulting Co., Ltd., Zhengzhou 450003, P. R. China

Abstract: The hydro unit economic load dispatch (ELD) is of great importance in energy conservation and emission reduction. Dynamic programming (DP) and genetic algorithm (GA) are two representative algorithms for solving ELD problems. The goal of this study was to examine the performance of DP and GA while they were applied to ELD. We established numerical experiments to conduct performance comparisons between DP and GA with two given schemes. The schemes included comparing the CPU time of the algorithms when they had the same solution quality, and comparing the solution quality when they had the same CPU time. The numerical experiments were applied to the Three Gorges Reservoir in China, which is equipped with 26 hydro generation units. We found the relation between the performance of algorithms and the number of units through experiments. Results show that GA is adept at searching for optimal solutions in low-dimensional cases. In some cases, such as with a number of units of less than 10, GA’s performance is superior to that of a coarse-grid DP. However, GA loses its superiority in high-dimensional cases. DP is powerful in obtaining stable and high-quality solutions. Its performance can be maintained even while searching over a large solution space. Nevertheless, due to its exhaustive enumerating nature, it costs excess time in low-dimensional cases.

Key words: hydro unit; economic load dispatch; dynamic programming; genetic algorithm; numerical experiment

1 Introduction

The economic load dispatch (ELD) is one of the major functions in automatic generation control (AGC), in which the objective is to meet the required load at the least cost through optimal load dispatch among units. It is of great importance in saving energy and reducing emission. Algorithms for solving ELD problems, including mathematical programming, stochastic optimization algorithms and hybrid algorithms (Bahmanifirouzi et al. 2012;
Hemamalini and Simon 2011; Somasundaram et al. 2006), have received increasing attention in recent years.

Most studies have developed conventional mathematical programming methods in early stages (Yamin 2004). Of these achievements, the equal lambda approach prevailed for a certain period of time due to its simplicity and computational advantages. However, it requires the cost function to be continuous and differentiable, and this requirement is hard to meet as the feasible range of unit power output is usually discontinuous considering the impact of the prohibited operation zone. Later, the Lagrangian relaxation (Benhamida and Abdelbar 2010; Ngundam et al. 2000) was introduced. It relaxes the load balance constraint using a Lagrange multiplier, and converges to the optimal solution through a multiplier-modifying procedure. Nevertheless, a duality gap (Ongsakul and Petcharak 2004) might block it from reaching the global optimum. Besides, both the initialization and iteration formulation have important roles in leading to convergence. In recent years, dynamic programming (DP) (Howard 1960) has become the most widely used algorithm for ELD. However, DP suffers from the curse of dimensionality; that is, its computation expense grows exponentially with the dimensionality of the problem. To avoid the curse of dimensionality, Liang and Glover (1992) applied incremental dynamic programming (IDP) to ELD. IDP reduces the size of required memory, but is unable to guarantee a global convergence. Yang (1995) proposed dimension-reduction dynamic programming that includes offline initialization and online dispatching. The initialization synthesizes the units of the same type to an equivalent virtual unit and provides the optimal cost curve for that virtual unit before the online dispatching. After that, the required load is dispatched among the virtual units first. Then, the dispatched load on the virtual units is reallocated to each real unit. A significant problem of DP and its modified versions is seeking a compromise between the solution quality and the computation expense.

Increasing numbers of studies have tried to use stochastic algorithms to solve ELD problem (Coelho and Mariani 2009; Sinha et al. 2003). The genetic algorithm (GA) (Holland 1975) is a commonly used stochastic algorithm due to its advantages in global optimization and parallelism. However, it has the deficiencies of poor local searching ability and premature convergence. In order to overcome these deficiencies, Hrstka and Kucerov (2004) proposed the concept of a radiation zone. Radiation zones are defined as adjacent regions of local optimums. If some chromosomes are trapped in these zones, their mutation probability will increase suddenly, which helps them to jump out the local optimum. Ongsakul and Ruangpayaongsak (2001) added a simulated annealing operator to GA to refine the elitist solutions and conducted numerical experiments to dispatch the load among 40 units. The results show that, although the quality of solutions has been improved, they require excessive computation time. Li et al. (1997) presented a hybrid GA combined with the gradient method, and validated its effectiveness in a 25-unit case.

Assessing the performance of each available method for solving the ELD problem is an
important issue. Although some studies have examined the performance of GA and DP, no consensus has been reached. Baskar et al. (2003) made a comparison between DP and a modified GA in a 10-unit system. The modified GA, which incorporates a shrinking solution space technique, outperforms DP in solution quality. Orero and Irving (1996) employed a binary code GA considering a crowd mechanism. The numerical results indicate that the modified GA is inferior in solution quality to DP for a 15-unit ELD problem. Bakirtzis et al. (1994) conducted numerical experiments in a system including 72 units. The results of the experiments show that GA is always superior in CPU time in all cases, but the convergence ratio and solution quality will decrease when the number of units surpasses 18. However, these conclusions necessitate further validation for the following reasons: First, the evaluation indexes in these studies were insufficient. Basically, they should at least include the following terms: CPU time, solution quality, and convergence ratio, which are used for assessing the computation expense, optimizing ability, and stability of algorithms, respectively. In addition, these experiments were not conducted in the same simulation environment (with the same experimental scheme, programming techniques, or computer hardware and software conditions), which is the major reason for the diversity of conclusions.

In this study, we formulated the ELD problem and used DP and GA, respectively, to solve it. For comparing the performances of the two algorithms, we introduced two experimental schemes and conducted numerical experiments. We applied the methodology to the Three Gorges Reservoir. Finally, we assessed the performance of DP and GA in different dimensions of the ELD problem through numerical experiments.

2 Hydro unit economic load dispatch

(1) Objective function:

The objective of ELD is to minimize the operating cost through optimal load dispatching among units. The operating cost for a hydropower station is usually evaluated by the power release. Therefore, the objective function can be expressed as

$$\min \; Q = \sum_{i=1}^{n} q_i (N_i, H)$$

(1)

(2) Constraints:

System load balance:

$$\sum_{i=1}^{n} N_i = N_d$$

(2)

Unit power output limits:

$$0 \leq N_i \leq N_{i_{\text{max}}} (H) \quad N_i \in [N_{i_{\text{min}}}, N_{i_{\text{max}}}] \quad j \in \Omega (H)$$

(3)

where $Q$ is the total power release from the reservoir ($\text{m}^3/\text{s}$); $n$ is the number of units; $q_i(\cdot)$ is the power release of turbine unit $i$ ($\text{m}^3/\text{s}$), and is specifically obtained through interpolation using the power output curve of unit $i$; $N_i$ is the power output of unit $i$ (MW); $H$ is the
average water head (m); \( N_d \) is the required load (MW); \( N_{i,\text{max}}(H) \) is the maximum power output of unit \( i \) at water head \( H \) (MW); \( N_{i,j} \) and \( N_{i,j}^\text{max} \) denote the lower bound and upper bound of power output of unit \( i \) in the \( j \)th prohibited zone of \( \Omega_i(H) \) (MW), respectively; and \( \Omega_i(H) \) is the set of prohibited zones of unit \( i \) at water head \( H \).

3 Methodology

3.1 Dynamic programming

By introducing the principle of optimality, DP converts a mathematical model into a multi-stage decision-making problem. In this study, DP included the following steps:

1. Determining stage and state variables:
   We selected the sequence number of unit \( (i) \) as the stage variable, and set the accumulated power output of units \( \sum_{i=1}^i N_i \) as the state variable. The accumulated power output is discretized with the increment of \( dN \):
   \[
   R_{ij} = \begin{cases} 
   \min\left(l dN, \sum_{i=1}^i M_i, N_d\right) & i \neq n \\
   N_d & i = n 
   \end{cases}
   \]
   \[
   l = 0, 1, \ldots, \text{int}\left(\min\left(\sum_{i=1}^i M_i, N_d\right) / dN\right) + 1
   \]
   (4)
   where \( R_{ij} \) is the value of the \( l \)th state variable at stage \( i \), \( M_i \) is the designed capacity of unit \( i \), and \( \text{int}() \) is the Gaussian rounding function.

2. Constraint handling:
   We used a penalty function to handle the power output constraint. It penalizes the objective function of infeasible solutions by adding extra discharge terms into the objective function:
   \[
   f_i(N_i,H) = q_i(N_i,H) + \Delta q_i + \Delta q_{pi}
   \]
   (5)
   \[
   \Delta q_i = \alpha_1 I_{NF}, \quad \Delta q_{pi} = \alpha_2 I_{NF}
   \]
   (6)
   \[
   \alpha_1 = \begin{cases} 
   1 & N_i \in [N_{l,i}, N_{u,i}], \exists j \\
   0 & N_i \notin [N_{l,i}, N_{u,i}], \forall j
   \end{cases}
   \]
   (7)
   \[
   \alpha_2 = \begin{cases} 
   1 & N_i \in (-\infty,0) \cup (N_{i,\text{max}},+\infty) \\
   0 & N_i \in [0,N_{i,\text{max}}]
   \end{cases}
   \]
   (8)
   where \( f_i(N_i,H) \) is the synthetic discharge of unit \( i \) at water head \( H \) (m³/s), \( \Delta q_i \) is the penalty term for increasing the discharge of unit \( i \) if its power output violates the prohibited zone constraint (m³/s), \( \Delta q_{pi} \) is another penalty term if the power output of unit \( i \) violates the power output limit constraint (m³/s), \( \alpha_1 \) and \( \alpha_2 \) are binary coefficients, and \( I_{NF} \) is an extremely large coefficient (m³/s).
(3) Recursive equation:
The changing of status from stage \(i-1\) to stage \(i\) can be characterized using the following equation:

\[
\sum_{i=1}^{i} N_i = \sum_{i=1}^{i-1} N_i + N_i
\]

(9)

The value of accumulated power release from unit 1 to \(i\) is updated using a forward recursive function

\[
Q_i' \left( \sum_{i=1}^{i} N_i \right) = \min \left( f_i (N_i, H) + Q_{i-1}' \left( \sum_{i=1}^{i-1} N_i \right) \right)
\]

(10)

where \(Q_i' \left( \sum_{i=1}^{i} N_i \right)\) is the optimal accumulated power release throughout past stages. The optimal solution can be found using a backward sweep from the last stage to the initial stage.

3.2 Genetic algorithm

To keep in accordance with DP in a similar solution space, an integer-coded GA was used in this study.

3.2.1 Constraints handling

The penalty function method (Chiang 2007; Kumar and Naresh 2009) and operator modification method (Michalewicz and Janikow 1996) are two conventional constraint-handling techniques. With the introduction of penalty functions, the original constrained optimization problem can be turned into an unconstrained problem. Unlike the penalty function method, the operator modification method is designed to make infeasible solutions feasible, which turns out to be a more efficient way of generating feasible solutions.

We combined the two techniques in the integer-coded GA. The operator modification method was used to handle the load balance constraint and unit power output constraint, while the penalty function method was used to deal with the prohibited zone constraint as given by Eq. (3).

3.2.2 Procedures

(1) Encoding and decoding:
We discretized the accumulated power output using Eq. (4) with an increment of \(dN'\).

\[
p_{k,i} \left( k = 1, 2, \cdots, P_{\text{op}}; i = 1, 2, \cdots, n \right) \text{ is defined as the } \theta \text{th gene of the } k\text{th chromosome, where } P_{\text{op}} \text{ is the population size. } p_{k,i} \text{ represents the column index of the accumulated power output matrix } R = (R_{ij}) \text{, and } p_{k,i} = 0, 1, \cdots, \text{int} \left( \min \left( \sum_{i=1}^{i} M_i, N_\text{a} \right) / dN' \right) + 1. \text{ } p_{k,i} \text{ can be used to get the accumulated power output of units 1 to } i \text{ from } R, \text{ as given by: } \sum_{i=1}^{i} N_i = R_{i,p_{k,i}}.
\]

(2) Chromosome initialization:
To ensure the efficiency of the algorithm in producing feasible solutions, we used a
heuristic method introduced by Zheng et al. (2013) to initialize the power output series. This method ensures that the load balance and power output limit constraints are strictly obeyed, using some linear transformation procedures during the initialization process.

(3) Fitness function:

The fitness function is used for evaluating the performance of a chromosome. Usually, a superior chromosome has a larger fitness function value. The fitness function $F_i$ is given by

$$F_i = \sum_{i=1}^{n} f_i \left( R_{i, j} - R_{i-1, j-1} + H \right)$$

(11)

(4) Crossover operator:

The arithmetic crossover is used and can be characterized as

$$p_{i, j}^* = \alpha p_{i, j} + (1 - \alpha) p_{i+1, j}, \quad p_{i, k+1}^* = (1 - \alpha) p_{i, j} + \alpha p_{i+1, j}$$

(12)

where $p_{i, j}$ and $p_{i+1, j}$ are the $i$th genes of the $i$th and $(i+1)$th parent chromosomes, respectively, and are selected to perform a crossover with the crossover probability $P_c$; $p_{i, k}$ and $p_{i, k+1}$ are the $i$th genes of the $k$th and $(k+1)$th child chromosomes, respectively; and $\alpha$ is a uniform random number in the range between 0 and 1.

(5) Mutation operator:

The mutation operator usually serves as a local search technique for the GA. A specific designed heuristic approach (Zheng et al. 2013) was used to facilitate the efficiency of GA in searching out the feasible space. The mutation probability is $p_m$.

(6) Selection operator:

We used the tournament selection to select those superior chromosomes from the entire chromosome set. After that, the selected ones were guided into the next generation, until the stopping criteria were met.

(7) Stopping criteria:

The algorithm stops when the number of generations in which the best chromosome remains unchanged is greater than a threshold ($S_{num}$) or the total number of generations reaches its expected value ($G_{en}$).

4 Numerical experimental schemes

4.1 Performance metrics

Three indices were selected for assessing the performances of DP and GA: the difference between objective functions, the difference between CPU times, and the convergence ratio. Note that the results of DP were selected as the benchmark, against which the differences were evaluated.

(1) Difference between objective functions:

We used the difference between the objective function values obtained by the GA and DP to assess the optimization ability, which is given by
\[ \Delta P_C = O_{GA} - O_{DP} \]  

where \( O_{DP} \) and \( O_{GA} \) are the optimal objective function values (in m\(^3\)/s) obtained by DP and GA, respectively. If \( \Delta P_C > 0 \), this demonstrates that DP outperforms the GA in solution quality. Otherwise, GA outperforms DP.

(2) Difference between CPU times:

\[ \Delta T = T_{GA} - T_{DP} \]  

where \( T_{DP} \) and \( T_{GA} \) are the CPU times (in s) of DP and GA, respectively.

(3) Convergence ratio of GA:

The convergence ratio is an index for illustrating the robustness of a stochastic algorithm. It equals the number of successful trials divided by the total number of trials:

\[ C_R = \frac{n_s}{n_t} \]  

where \( n_t \) is the total number of trials, and \( n_s \) is the number of successful trials. When the optimal solution of GA is close to any one of optimal solutions of DP within a given tolerance, it is defined as a successful trial:

\[ | N_{GA} - N_{DP} | \leq \varepsilon \quad N_{DP} \in \mathcal{Y}_{DP} \]  

where \( N_{DP} \) is one of the optimal solutions obtained by DP and recorded in the optimal solution set of DP \( \mathcal{Y}_{DP} \), \( N_{GA} \) is the optimal solution obtained by GA, and \( \varepsilon \) is the tolerance.

### 4.2 Steps for numerical experiments

In general, the quality of the optimal solution obtained by an algorithm can be improved by increasing the CPU time. For instance, GA may obtain a better solution if a larger population size is provided or a larger number of generations is given. Likewise, DP can obtain a better solution on the condition that the solution space is discretized with a smaller increment. Therefore, in consideration of the fairness principle, evaluation of simulations involving an efficiency comparison should concern both the solution quality and CPU time (Cheng et al. 2009; Panigrahi et al. 2008). Accordingly, we established two schemes:

(1) Scheme A: comparing the CPU times required to obtain identical solutions.

We set the increment of GA equal to that of DP, so that the two algorithms had the same solution spaces. We then solved the problem using DP, and adjusted GA parameters to obtain the same solution as what DP found. Since the population size was the most sensitive factor of GA (Zhong and Tang 2010), we calibrated it carefully by introducing some predefined criteria. After we solved the same model using a calibrated GA, we compared the difference between CPU times.

(2) Scheme B: comparing the solution quality values within the same CPU time.

We solved the ELD problem with DP and recorded its CPU time. Then, we controlled the CPU time of GA; that is, we stopped GA when it consumed the same amount of time as DP.
Note that the increment of GA in scheme B needed to be smaller than that in scheme A. Otherwise, GA could not obtain a better solution than DP.

5 Case study

Numerical experiments were conducted using the case of the Three Gorges Reservoir in China. The Three Gorges Reservoir, which is located in the middle reach of the Yangtze River, is one of the largest water conservancy projects in the world. It has an effective storage capacity of $1.65 \times 10^{10}$ m$^3$ in order to regulate an average annual runoff of $4.510 \times 10^{11}$ m$^3$. The Three Gorges Reservoir has 26 hydro units of five different types, and generates $8.47 \times 10^7$ MW·h of electric power annually. The characteristics of the Three Gorges Reservoir are given in Table 1.

<table>
<thead>
<tr>
<th>Powerhouse</th>
<th>Unit type</th>
<th>Number of units</th>
<th>Maximum discharge per unit (m$^3$/s)</th>
<th>Designed capacity per unit (MW)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Left bank</td>
<td>VGS</td>
<td>6</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ALSTOM I</td>
<td></td>
<td>8</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Dongfang</td>
<td></td>
<td>4</td>
<td>996</td>
<td>700</td>
</tr>
<tr>
<td>Right bank</td>
<td>ALSTOM II</td>
<td>4</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Harbin</td>
<td></td>
<td>4</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

We set the increments of both DP and GA to be 14 MW for scheme A. For scheme B, the increment of GA was set to be 1.4 MW. In addition, parameters determining GA's performance were optimized with the trial and error method. As a result, the crossover probability $P_c$ was 1, the mutation probability $P_m$ was 0.1, and the convergence threshold $S_{\text{num}}$ was 5. We set the expected number of generations $G_{\text{en}}$ at 100, and the tolerance $\epsilon$ at $2dN\sqrt{n}$. Other parameters associated with different schemes were set as follows:

1. For scheme A, the lower bound of the convergence ratio $C_{Re}$ was set at 40%.
2. For scheme B, the population size satisfied the condition $P_{op} = 5n$.

5.1 Model validation

We selected a typical day from the historical records of the Three Gorges Reservoir. We conducted two set of experiments, where the required load $N_d$ was set as the peak load and off-peak load of that day, respectively. We ran the model for each of the required loads to find the optimal dispatch. Table 2 lists the results. The mean CPU times of DP and GA are 213.5 s and 169.3 s, respectively.

The simulated results show the following:

1. For the off-peak load, both DP and GA have reached the same objective function value (6331 m$^3$/s), but the solutions of DP and GA are not the same. There are multiple optimal solutions to the ELD problem.
Table 2: Optimal load dispatch for a peak and off-peak load of a typical day

<table>
<thead>
<tr>
<th>Unit ID</th>
<th>Off-Peak load</th>
<th>Peak load</th>
<th>Off-Peak load</th>
<th>Peak load</th>
<th>Off-Peak load</th>
<th>Peak load</th>
<th>Off-Peak load</th>
<th>Peak load</th>
<th>Off-Peak load</th>
<th>Peak load</th>
</tr>
</thead>
<tbody>
<tr>
<td>VGS 1#</td>
<td>0</td>
<td>560</td>
<td>0</td>
<td>604</td>
<td>0</td>
<td>672</td>
<td>0</td>
<td>713</td>
<td></td>
<td></td>
</tr>
<tr>
<td>VGS 2#</td>
<td>0</td>
<td>560</td>
<td>0</td>
<td>604</td>
<td>0</td>
<td>560</td>
<td>0</td>
<td>604</td>
<td></td>
<td></td>
</tr>
<tr>
<td>VGS 3#</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>VGS 4#</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>VGS 5#</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>VGS 6#</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ALSTOM I 1#</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ALSTOM I 2#</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ALSTOM I 3#</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ALSTOM I 4#</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ALSTOM I 5#</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ALSTOM I 6#</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ALSTOM I 7#</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ALSTOM I 8#</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Dongfang 1#</td>
<td>686</td>
<td>0</td>
<td>722</td>
<td>0</td>
<td>700</td>
<td>700</td>
<td>735</td>
<td>735</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Dongfang 2#</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>700</td>
<td>0</td>
<td>735</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Dongfang 3#</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Dongfang 4#</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Harbin 1#</td>
<td>700</td>
<td>700</td>
<td>735</td>
<td>735</td>
<td>700</td>
<td>700</td>
<td>735</td>
<td>735</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Harbin 2#</td>
<td>700</td>
<td>686</td>
<td>735</td>
<td>722</td>
<td>700</td>
<td>574</td>
<td>735</td>
<td>618</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Harbin 3#</td>
<td>700</td>
<td>672</td>
<td>735</td>
<td>709</td>
<td>700</td>
<td>0</td>
<td>735</td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Harbin 4#</td>
<td>700</td>
<td>672</td>
<td>735</td>
<td>709</td>
<td>700</td>
<td>0</td>
<td>735</td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ALSTOM II 1#</td>
<td>632</td>
<td>630</td>
<td>669</td>
<td>667</td>
<td>686</td>
<td>658</td>
<td>720</td>
<td>694</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ALSTOM II 2#</td>
<td>630</td>
<td>630</td>
<td>667</td>
<td>667</td>
<td>630</td>
<td>630</td>
<td>667</td>
<td>667</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ALSTOM II 3#</td>
<td>630</td>
<td>630</td>
<td>667</td>
<td>667</td>
<td>602</td>
<td>602</td>
<td>641</td>
<td>641</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ALSTOM II 4#</td>
<td>630</td>
<td>630</td>
<td>667</td>
<td>667</td>
<td>590</td>
<td>574</td>
<td>630</td>
<td>615</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Sum</td>
<td>6008</td>
<td>6370</td>
<td>6331</td>
<td>6749</td>
<td>6008</td>
<td>6370</td>
<td>6331</td>
<td>6756</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

(2) For the peak load, although GA obtains a larger objective function value (6 756 m³/s) than DP (6 749 m³/s), GA consumes less CPU time. Therefore, there can be a tradeoff between solution quality and CPU time. To analyze the efficiency of algorithms, we have to conduct comparisons systematically through numerical experiments.

5.2 Results from numerical experiments

We ran the model repetitively in the Windows XP operating system using Visual Basic 6.0 programs. The computer was equipped with an Intel Core 2 Duo E7500 CPU and 2 GB of RAM. The results are listed in Table 3 and through Fig. 1 to Fig. 3.
Table 3 Results of performance metrics from numerical experiments

<table>
<thead>
<tr>
<th>Number of units</th>
<th>Mean load (MW)</th>
<th>Mean CPU time (s)</th>
<th>Mean power release (m³/s)</th>
<th>Scheme A</th>
<th>GA</th>
<th>Scheme B</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Mean power release (m³/s)</td>
<td>$P_{op}$</td>
<td>Mean $\Delta T$ (s)</td>
</tr>
<tr>
<td>2</td>
<td>1 278</td>
<td>0.011</td>
<td>1 394</td>
<td>1 394</td>
<td>35</td>
<td>0.203</td>
</tr>
<tr>
<td>4</td>
<td>2 343</td>
<td>1.132</td>
<td>2 553</td>
<td>2 553</td>
<td>35</td>
<td>−0.541</td>
</tr>
<tr>
<td>6</td>
<td>3 513</td>
<td>3.667</td>
<td>3 824</td>
<td>3 824</td>
<td>50</td>
<td>−1.900</td>
</tr>
<tr>
<td>8</td>
<td>4 749</td>
<td>7.650</td>
<td>5 151</td>
<td>5 152</td>
<td>90</td>
<td>−3.274</td>
</tr>
<tr>
<td>10</td>
<td>5 958</td>
<td>13.103</td>
<td>6 447</td>
<td>6 447</td>
<td>110</td>
<td>−1.563</td>
</tr>
<tr>
<td>12</td>
<td>6 429</td>
<td>19.060</td>
<td>6 962</td>
<td>6 965</td>
<td>150</td>
<td>4.180</td>
</tr>
<tr>
<td>14</td>
<td>7 450</td>
<td>27.213</td>
<td>8 054</td>
<td>8 057</td>
<td>150</td>
<td>1.367</td>
</tr>
<tr>
<td>16</td>
<td>8 134</td>
<td>35.285</td>
<td>8 804</td>
<td>8 809</td>
<td>165</td>
<td>3.948</td>
</tr>
<tr>
<td>18</td>
<td>9 119</td>
<td>45.577</td>
<td>9 852</td>
<td>9 855</td>
<td>200</td>
<td>7.251</td>
</tr>
<tr>
<td>20</td>
<td>11 107</td>
<td>58.749</td>
<td>1 2003</td>
<td>12 006</td>
<td>210</td>
<td>3.151</td>
</tr>
<tr>
<td>22</td>
<td>10 793</td>
<td>68.558</td>
<td>1 1675</td>
<td>11 687</td>
<td>250</td>
<td>11.072</td>
</tr>
<tr>
<td>24</td>
<td>10 816</td>
<td>77.855</td>
<td>1 1682</td>
<td>11 680</td>
<td>250</td>
<td>22.645</td>
</tr>
<tr>
<td>26</td>
<td>12 366</td>
<td>95.230</td>
<td>1 3350</td>
<td>13 357</td>
<td>275</td>
<td>27.309</td>
</tr>
</tbody>
</table>

From the results, the following can be inferred:

1. According to Fig. 1, given the condition that the two algorithms acquire identical solutions, GA outperforms DP in CPU time when the number of units is less than 10. Otherwise, the results are reversed.

2. As shown in Fig. 2, on the condition that the two algorithms consume the same CPU time, GA can obtain a better solution than DP if the number of units is less than or equal to 4.
However, the performance of GA is inferior to that of DP with the increase of the number of units, especially when the number of units is greater than 10.

(3) As Fig. 3 shows, the convergence ratio of GA decreases with the increase of the number of units in both schemes. This is because the prematurity problem limits the performance of GA.

As discussed above, the performance of an algorithm can be improved if more CPU time is given. For instance, DP can find a better solution if we use a smaller increment while GA can improve its solution quality as well if a larger population size is used. For the conditions and parameters discussed here, it is clear that both DP and GA can perform well in either CPU time or solution quality. There then exists a tradeoff between CPU time and solution quality, which helps to identify how much computation is necessary to improve the solution quality. Therefore, we conducted a perturbation analysis to obtain the tradeoff by varying the major parameters of GA and DP. The relationships between the power release and CPU time for different numbers of units are shown in Fig. 4.

![Fig. 4 Power release vs. CPU time for different numbers of units](image)

Note that we fit the scatters in Fig. 4 with exponential curves. Fig. 4 shows that the solution quality can be improved for both algorithms with the increase of CPU time, but the improvement decreases gradually. This rule demonstrates that it is not worthwhile to pursue a minor improvement at the expense of consuming a large amount of CPU time.

GA has an advantage over DP in finding better solutions for solving low-dimensional problems when the CPU time is very limited. For instance, for solving an ELD problem for a 16-unit system, GA is not as efficient as DP. This is because GA is a heuristic search method; it will lose its efficiency when finding a global optimum in a large solution space. Also, GA's
deficiency in performing local searches limits its performance in refining solution quality. In contrast, DP searches the solution space thoroughly by enumerating the discretized grids. Although it costs significant CPU time, the solution quality can be guaranteed.

6 Conclusions

In this study, we conducted numerical experiments to compare the efficiency of DP and GA in hydro unit economic load dispatch. The experiments were applied to the Three Gorges Reservoir, which has 26 hydro units. Specifically, we introduced the performance metrics and designed two experimental schemes for conducting a performance comparison. The results of the experiments show the following:

(1) The performances of the two algorithms are strongly affected by their parameters as well as their applications. GA is adept at searching for optimal solutions in low-dimensional cases. In some cases, such as with a number of units of less than 10, the performance of GA is superior to that of DP. However, as a heuristic search method, GA loses its superiority in high-dimensional cases.

(2) DP has a good performance in obtaining stable and high-quality solutions, even in searching through large solution space. However, due to its exhaustive enumerating nature, it costs excess time in low-dimensional cases.

References


(Edited by Yan LEI)