GENETIC ALGORITHM BASE REACTIVE POWER DISPATCH AND VOLTAGE CONTROL

Sunil P. Patel

1Electrical Department, LDRP-ITR (Kadi sarva vidhalya uni.), Gandhinagar, Gujarat, India, patelsunil33@yahoo.com

Abstract - Reactive power control becomes extremely important after deregulation because customers request high quality power supply with reasonable price. Also the Voltage profile at different nodes in the power system is affected greatly by the variations in load and generation profiles during normal and abnormal operating states. The reactive power control devices such as generators, tap positions of on-load tap changer of transformers, shunt reactors are used to correct voltage limits violations while simultaneously reducing the system real power losses. Genetic algorithms (GAs) are well-known global search techniques anchored on the mechanisms of natural selection and genetics. Here I present the work on genetic algorithm for reactive power optimization and voltage control. The objective of genetic algorithm base reactive power optimization is minimizing active power losses while maintaining the quality of voltages.


I. INTRODUCTION

Carpentier first defined the OPF problem in early 1960s and OPF soon many researchers. Optimal power flow (OPF) is one of the main functions of power generation operation and control. Power flow study aims at reaching to the steady state solution of complete power networks. Power flow study is performed during the planning of a new system or the extension of an existing system. It is also necessary to evaluate the effect of different loading conditions of an existing system. Power flow equations represent a set of non-linear simultaneous algebraic equations, for which there has been no general solution until now. OPF is a nonlinear, non-convex, large-scale, static optimization problem with both continuous and discrete control variables. Even in the absence of discrete control variables, the OPF problem is non-convex due to the existence of the nonlinear (AC) power flow equality constraints [12].

II. PROBLEM FORMULATION

The formula for optimal power flow is represented by following standard form,

Minimize \( f(u,x) \)

Subject to

\[ g(u,x) = 0 \]

\[ h(u,x) \leq 0 \]

The objective function \( f(u,x) \) represent the system optimization goal. \( f \) is usually a scalar function, but in multi objective OPF it may be interpreted as a vector function. Vector function \( g(u,x) \) and \( h(u,x) \) represent system equality and inequality constraints respectively.

The objective of the reactive power optimization is to minimize the active power losses of the system, which is defined as follows [1].

\[
P_{loss} = f(\vec{x}_1, \vec{x}_2)
\]

\[
P_{loss} = \sum_{k \in N_E} g_k (V_i^2 + V_j^2 - 2V_iV_j \cos \theta_{ij})
\]

In above equation \( f(\vec{x}_1, \vec{x}_2) \) denotes the active power loss function of the transmission network, \( \vec{x}_1 \) is the control variable vector \([V_G, T_K, Q_C]\), \( \vec{x}_2 \) is the dependent variable vector \([V_L, Q_G]\), \( V_G \) is the generator voltage.
(continuous), \( T_K \) is the transformer tap (integer), \( Q_C \) is the shunt capacitor or reactor (integer), \( V_l \) is the load bus voltage, \( Q_G \) is the generator reactive power, \( k = (i, j) \), \( i \in N_B \) and \( j \in N_i \), \( g_k \) is the conductance of branch \( k \), \( \theta_{ij} \) is the voltage angle difference between bus \( i \) and \( j \), \( N_E \) is the set of number of network branches.

Equality constrain [1],

\[
P_{G_i} - P_{D_i} - V_i \sum_{j \in N_i} V_j (G_{ij} \cos \theta_{ij} + B_{ij} \sin \theta_{ij}) Q_{G_i} - Q_{D_i} - V_i \sum_{j \in N_i} V_j (G_{ij} \sin \theta_{ij} - B_{ij} \cos \theta_{ij})
\]

The inequality constraints on control (independent) variable limits are described as follows,

Inequality constrain [1],

\[
V_{i_{\min}} \leq V_i \leq V_{i_{\max}} \quad i \in N_B
\]

\[
T_{k_{\min}} \leq T_k \leq T_{k_{\max}} \quad K \in N_T
\]

\[
Q_{G_{i_{\min}}} \leq Q_{G_i} \leq Q_{G_{i_{\max}}} \quad i \in N_G
\]

\[
Q_{C_{i_{\min}}} \leq Q_{C_i} \leq Q_{C_{i_{\max}}} \quad i \in N_C
\]

\[
S_l \leq S_{l_{\max}} \quad l \in N_l
\]

In these constraints \( P_{G_i} \) and \( Q_{G_i} \) are the injected active and reactive power at bus \( i \), \( P_{D_i} \) and \( Q_{D_i} \) are the demanded active and reactive power at bus \( i \), \( G_{ij} \) and \( B_{ij} \) are the transfer conductance and transfer susceptance between bus \( V \) and \( J \), \( N_{PQ} \) is the set of numbers of PQ buses, \( N_B \) is the set of total numbers of buses, \( N_i \) is the set of numbers of buses adjacent to bus \( i \) including bus \( i \), \( N_o \) is the set of numbers of buses excluding slack bus, \( N_c \) is the set of numbers of possible reactive power sources installation buses, \( N_g \) is the set of numbers of generator buses, \( N_T \) is the set of numbers of transformer branches, \( S_l \) is the power flow in branch \( l \).

### III. METHODOLOGY

**A. Initialization of Population**

Population is a matrix having strings of binary numbers i.e. 0 and 1 which are nothing except potential solution to the problem. We generate a random population to start the solution. A set of real-coded initial populations are generated randomly within the minimum and maximum limits of the control variables and it is chosen as the parent population.

\[
\text{chromosome} = \text{round} \left( \text{rand(pop\_size,nbit)} \right);
\]

\[
\text{chromosome} = \text{randi} \left( [0,1], \text{pop\_size}, \text{nbit} \right);
\]

Where

- pop\_size = number of strings or simply you can number of potential solutions
- nbit = it is multiplication of bits per variable and number of variables.

As these binary strings must be converted into real numbers, this is done by using Equation

\[
P_i = P_i^{lower} + \frac{P_i^{upper} - P_i^{lower}}{2^n - 1} \times B_i^{decld}
\]
Where

\[ p_i = \text{Real limit of that variable} \]
\[ p_{i,\text{lower}} = \text{lower limit of that variable} \]
\[ p_{i,\text{upper}} = \text{upper limit of that variable} \]
\[ p_{i,\text{decd}} = \text{binary to decimal converted value of that variable} \]

**B. Function Calculation**

After decoding all the strings of the initial population and get values of each variable, we put them into our function and get its values.
C. Fitness Calculation
Fitness is the ability of a particular string to solve the problem. It is also a way for satisfying equality constraints. Implementation of a problem in genetic algorithm is realized within the fitness function and in order to emphasize the ‘best’ chromosomes and speed up convergence of the iteration procedure, fitness is normalized into the range between 0 and 1. Fitness calculation is done in following steps.

Step 1. We first calculate error, which is nothing but our equality constraint.
Step 2. Fitness function is \( \text{fitn}_i = \frac{1}{\text{function}} \).

D. Sorting
It gives search direction to the program by sorting fitness in descending order and according to that order it sorts initial population, error and function values.

E. Selection
Roulette wheel selection procedure uses fitness of that string to make slot sizes.

Step 1. Calculate fitness of each string.
Step 2. Calculate probability of each string i.e. \( \text{prob}_i = \frac{\text{fitn}_i}{\sum_{j=1}^{n} \text{fitn}_j} \), where \( n \) is population size.
Step 3. Calculate cumulative probability for each string by using following equation \( \text{comprob}_i = \sum_{j=1}^{n} \text{prob}_j \)
Step 4. Generate random number \( 0 < r < 1 \)
Step 5. If \( r < \text{comprob}_i \) then select the first string, or select the string which has less value of cumulative probability than randomly generated number.
Step 6. Repeat Step 4 and 5 for number of strings is needed.

The whole procedure is explained with an example. Suppose we have \( \text{pop}_\text{size} = 5 \). The total fitness \( \sum_{i=1}^{5} \text{fitn}_i = 15 + 20 + 27 + 08 + 23 = 93 \). The probability of selecting individual string, its cumulative probability and fitness is shown in following Table-1.

<table>
<thead>
<tr>
<th>Sting</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fitness</td>
<td>15</td>
<td>20</td>
<td>27</td>
<td>08</td>
<td>23</td>
</tr>
<tr>
<td>Probability</td>
<td>15/93 = 0.1612</td>
<td>0.2150</td>
<td>0.2903</td>
<td>0.08602</td>
<td>0.2473</td>
</tr>
<tr>
<td>Cumulative Probability</td>
<td>0.1612</td>
<td>0.3762</td>
<td>0.6665</td>
<td>0.7525</td>
<td>1</td>
</tr>
</tbody>
</table>

Table 1 -Roulette Wheel selection method

F. Crossover and Mutation
Selected strings, now onwards we will call them parents, are used to do crossover and mutation to produce children which are responsible for the global search property of the GA. Crossover operator basically combines substructures of two parent chromosome to produce new structure which is child. There are many types of crossover like single point, double point, multipoint, uniform, matrix etc. The final operator in the genetic algorithm is mutation; the mutation operator is used to inject new information into the population. Mutation changes randomly the new offspring. For binary
mutation is preferred which switches a few randomly chosen bits from 1 to 0 and vice-a-versa. There are many types of mutation processes like uniform, boundary, non-uniform etc. After mutation, the new generation is completed and the procedure begins again with the function calculation of the population and so on.

\[
\text{chromosome} = \\
\begin{array}{cccccc}
0 & 1 & 1 & 1 & 0 & 1 \\
0 & 0 & 0 & 1 & 1 & 1 \\
0 & 0 & 1 & 0 & 1 & 1
\end{array}
\]

Original population

Single Point CrossOver

\[
\text{chromosome} = \\
\begin{array}{cccccc}
0 & 1 & 1 & 0 & 1 & 1 \\
0 & 0 & 0 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 & 0 & 0
\end{array}
\]

3rd string

Cross over point is at 3.

1st string 2nd string

\[
\text{chromosome} = \\
\begin{array}{cccccc}
0 & 1 & 1 & 0 & 1 & 1 \\
0 & 0 & 0 & 0 & 0 & 0 \\
0 & 1 & 1 & 0 & 0 & 0
\end{array}
\]

4th string crossover point is at 3.

2nd string 1st string

Random Bitwise Mutation

mutrow = 
3 4

mutcol =
4 3

\[
\text{chromosome} = \\
\begin{array}{cccccc}
0 & 1 & 1 & 0 & 1 & 1 \\
0 & 0 & 0 & 0 & 0 & 0 \\
0 & 1 & 1 & 1 & 0 & 0 \\
0 & 0 & 0 & 0 & 1 & 1
\end{array}
\]

Bit number (3,4) was originally 0 and mutate to 1.

\[
\text{chromosome} = \\
\begin{array}{cccccc}
0 & 1 & 1 & 0 & 1 & 1 \\
0 & 0 & 0 & 0 & 0 & 0 \\
0 & 1 & 1 & 0 & 0 & 0 \\
0 & 0 & 1 & 0 & 1 & 1
\end{array}
\]

Bit number (4,3) was originally 0 and mutate to 1.

IV. SIMULATION RESULT

<table>
<thead>
<tr>
<th>Variable</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>V1(p.u)</td>
<td>1.0221</td>
</tr>
<tr>
<td>V2(p.u)</td>
<td>1.0190</td>
</tr>
<tr>
<td>V5(p.u)</td>
<td>1.0182</td>
</tr>
</tbody>
</table>
Table – 2 shows the results of MATLAB program on IEEE 30-bus test system.

<table>
<thead>
<tr>
<th>Variables</th>
<th>Value using GA</th>
<th>Value using NR load flow</th>
</tr>
</thead>
<tbody>
<tr>
<td>V8 (p.u.)</td>
<td>0.9888</td>
<td></td>
</tr>
<tr>
<td>V11 (p.u.)</td>
<td>0.9752</td>
<td></td>
</tr>
<tr>
<td>V13 (p.u.)</td>
<td>0.9803</td>
<td></td>
</tr>
<tr>
<td>Pgen1 (MW)</td>
<td>1.4570</td>
<td></td>
</tr>
<tr>
<td>Pgen2 (MW)</td>
<td>0.2584</td>
<td></td>
</tr>
<tr>
<td>Pgen3 (MW)</td>
<td>0.1732</td>
<td></td>
</tr>
<tr>
<td>Pgen1 (MW)</td>
<td>0.1340</td>
<td></td>
</tr>
<tr>
<td>Pgen1 (MW)</td>
<td>0.1784</td>
<td></td>
</tr>
<tr>
<td>Pgen1 (MW)</td>
<td>0.1263</td>
<td></td>
</tr>
<tr>
<td>Tap 9</td>
<td>0.9130</td>
<td></td>
</tr>
<tr>
<td>Tap 10</td>
<td>1.0935</td>
<td></td>
</tr>
<tr>
<td>Tap 12</td>
<td>0.9661</td>
<td></td>
</tr>
<tr>
<td>Tap 27</td>
<td>0.9600</td>
<td></td>
</tr>
<tr>
<td>Qsh 10</td>
<td>0.2229</td>
<td></td>
</tr>
<tr>
<td>Qsh 24</td>
<td>0.2390</td>
<td></td>
</tr>
</tbody>
</table>

Table – 3 shows comparison of optimal values of variables generated using genetic algorithm and values of the same variables in normal load flow analysis using NR method.

<table>
<thead>
<tr>
<th>Variables</th>
<th>Value using GA</th>
<th>Value using NR load flow</th>
</tr>
</thead>
<tbody>
<tr>
<td>V1 (p.u.)</td>
<td>1.0175</td>
<td>1.06</td>
</tr>
<tr>
<td>V2 (p.u.)</td>
<td>1.0182</td>
<td>1.045</td>
</tr>
<tr>
<td>V3 (p.u.)</td>
<td>1.0035</td>
<td>1.01</td>
</tr>
<tr>
<td>V6 (p.u.)</td>
<td>0.9968</td>
<td>1.07</td>
</tr>
<tr>
<td>V8 (p.u.)</td>
<td>1.0038</td>
<td>1.08</td>
</tr>
<tr>
<td>Pgen1 (MW)</td>
<td>21.065</td>
<td>0</td>
</tr>
<tr>
<td>Pgen2 (MW)</td>
<td>70</td>
<td>40</td>
</tr>
<tr>
<td>Pgen3 (MW)</td>
<td>50</td>
<td>0</td>
</tr>
<tr>
<td>Pgen6 (MW)</td>
<td>70</td>
<td>0</td>
</tr>
<tr>
<td>Pgen8 (MW)</td>
<td>50</td>
<td>0</td>
</tr>
<tr>
<td>Tap8</td>
<td>0.998</td>
<td>0.978</td>
</tr>
<tr>
<td>Tap9</td>
<td>0.989</td>
<td>0.969</td>
</tr>
<tr>
<td>Tap10</td>
<td>0.995</td>
<td>0.932</td>
</tr>
</tbody>
</table>

V. CONCLUSION

The proposed Genetic algorithm approach to obtain the optimum values of the reactive power Variables including the voltage stability constraint. The effectiveness of the proposed method is demonstrated on IEEE-14 and IEEE-30 bus system with promising results. The performance of the proposed algorithm is demonstrated through its voltage stability
enhancement by simulation. From this multi-objective reactive power optimization solution, the application of GA has performed well when it was used to characterize Pareto optimal front and leads to global search with fast convergence rate and a feature of robust computation. From the simulation work, it is concluded that GA performs better results than other evolutionary methods.

VI. REFERENCE


