GENETIC ALGORITHM BASED LOAD FLOW SOLUTION PROBLEM IN ELECTRICAL POWER SYSTEMS

By

Asst. Prof. Hassan A. Kubba
Electrical Engineering Department
Baghdad University

Samir Sami Mahmood
M.Sc. Electrical Engineering
Baghdad University

ABSTRACT

In this paper, a proposed method based on real-coded genetic algorithm is presented and applied to solve multiple load flow solution problem. Genetic algorithm is a kind of stochastic search algorithm based on the mechanics of natural selection and natural genetics. They combine the concepts of survival of the fittest with genetic operators such as selection, crossover and mutation abstracted from nature to form a surprisingly robust mechanism that has been successfully applied to solve a variety of search and optimization problems. Elitist method is also used in this research, and blending models are implemented for crossover operator. In the proposed work, five busbars typical test system and 362-bus Iraqi National Grid are used to demonstrate the efficiency and performance of the proposed method. The results show that, genetic algorithm is on-line load flow solution problem for small-scale power systems, but for large-scale power systems, it is recommended that the load flow solution using genetic algorithm is for planning studies. The main important feature of the purposed method is to give high accurate solution with respect to the conventional methods.

الخلاصة

في هذه البحث، تقدم طريقة مقترحة مبنية على اساس خوارزمية جينية مشفرة بالاعداد الحقيقية لحل مسألة سريان الحمل متعددة الحلول. تعتبر الخوارزمية الجينية احدى طرق البحث العشوائية القائمة على تقنيات الانتخاب (الانتقاء) الطبيعي و الجينات الطبيعية. تجمع الخوارزمية الجينية مبادئ (بقاء الاصلح) مع عوامل جينية كالانتخاب (الانتقاء)، العبور و التغيار الاحيائي (الطفرة) المستخلصة من الطبيعة لتكوين تقنية متينة استخدمت بنجاح في حل مختلف مسائل البحث و ايجاد القيم المثلى. تم في هذا البحث استخدام طريقة انتخاب الامثل بالاضافة الى استخدام نماذج الخلط في عملية العبور. لتبيان كفاءة و مدى فعالية الخوارزمية الجينية في حل مسائل سريان الحمل متعددة الحلول، تم تطبيق الطريقة المقترحة على منظومة قدرة كهربائية قياسية. تبين النتائج كون الطريقة المقترحة ملائمة للحل اللحظي لمسائل سريان الحمل و بالتالي التطبيق العملي اثناء التشغيل لمنظومات القدرة صغيرة الحجم. أما بالنسبة لمنظومات القدرة كبيرة الحجم، فيوصي الباحث باستخدام الطريقة المقترحة لاغراض التصميم و التخطيط. اهم خواص الطريقة المقترحة هي الحصول على نتائج و حلول لمسألة سريان الحمل بقيم عالية الدقة.

KEYWORDS

Continuous Genetic Algorithm, Chromosome Crossover, Load Flow Analysis, Newton-Raphson Method, Mutation, Multi-Objective Minimization.

INTRODUCTION

With increasing computer speeds, researchers are increasingly applying artificial and computational intelligence techniques, especially in power system problems. These methods offer several advantages over traditional numerical methods. Among these techniques is that of genetic algorithm. Genetic algorithms (GAs) are efficient stochastic search algorithms that emulate natural phenomena. They have been used successfully to solve wide range of optimization problems. Because of existence of local optima, these algorithms offer promise in solving large-scale problems. A genetic algorithm mimics Darwin's evolution process by implementing "survival of the fittest" strategy. Genetic algorithm solves linear and nonlinear problems by exploring all regions of the search space and exponentially exploiting promising areas through selection, crossover, and mutation operations. They have been proven to be an effective and flexible optimization tool that can find optimal or near-optimal solutions [Talib 2007]. In this study, an improved genetic algorithm solution of the load flow problem is presented in order to minimize the total active and reactive power mismatches of the given systems, a real-coded genetic algorithm has been implemented.

THE CONCEPTS OF LOAD FLOW ANALYSIS

The load flow studies are the backbone of the design of a power system. They are the means by which the future operation of the system is known ahead of time. The load flow problem is one of the basic problems in the power system engineering, and can be expressed as a set of non-linear simultaneous algebraic equations, and thus it is to have multiple solutions [Woon 2004]. A load flow study is the determination of voltage, current, power, and power factor or reactive power at various points in an electrical network under existing or contemplated conditions of normal operation, so power flow calculations provide power flows and voltages for a specified power system subject to the regulating capability of generators, condensers, and tap changing under load transformers as well as specified net interchange between individual operating systems. This information is essential for the continuous evaluation of the current performance of a power system and for analyzing the effectiveness of alternative plans for system expansion to meet increased load demand. The continual expansion of the demand for electrical energy due to the growth of industries, commercial centers, and residential sections requires never-ending additions to existing power systems. The systems engineer must decide what components must be added to the system many years before they are put into operation and he does this by means of power flow studies. The load flow solution usually provides additional information, e.g. losses [Kubba 1987]. The load flow is the most frequently carried out study by power utilities and is required to be performed at almost all the stages of power system planning, optimization, operation, control, and contingency analysis.

ITERATIVE NUMERICAL (LF) SOLUTION METHODS

* NEWTON-RAPHSON METHOD

At each iteration of the Newton-Raphson method, the nonlinear problem is approximated by a linear matrix equation (Jacobian matrix). The linearzing approximation can best be visualized in the case of a single-variable problem as shown in figure (1).

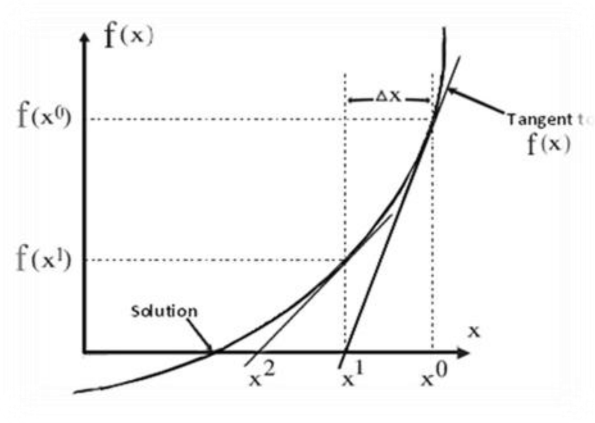


Fig. 1 Single-Variable Linear Approximation [Taylor 1967]

The Newton-Raphson load flow equation are

$$\Delta P_k = P_k{}^{sp} - |V_k| \sum_{m=1}^{N} |V_m| \left(g_{km}\cos\delta_{km} + b_{km}\sin\delta_{km}\right)$$

$$\Delta Q_k = Q_k{}^{sp} - |V_k| \sum_{m=1}^{N} |V_m| \left(g_{km}\sin\delta_{km} - b_{km}\cos\delta_{km}\right)$$

where $\begin{array}{l}\bar{V}_k = |V_k|\, e^{j\delta_k} \\ \bar{V}_m = |V_m|\, e^{j\delta_m}\end{array}$, $\begin{array}{l}\bar{I}_k = |I_k| e^{j\alpha_k} \\ \bar{Y}_{km} = |Y_{km}| e^{j\theta_{km}} = g_{km} + jb_{km}\end{array}$ and $\delta_{km} = \delta_k - \delta_m$ (Kubba 1987).

The Jacobian matrix equation can be written as:

$$\begin{bmatrix} J_1 & J_2 \\ J_3 & J_4 \end{bmatrix} \begin{bmatrix} \Delta\delta_1 \\ \Delta\delta_2 \\ \vdots \\ \Delta\delta_N \\ \Delta V_1 \\ \Delta V_2 \\ \vdots \\ \Delta V_N \end{bmatrix} = - \begin{bmatrix} \Delta P_1 \\ \Delta P_2 \\ \vdots \\ \Delta P_N \\ \Delta Q_1 \\ \Delta Q_2 \\ \vdots \\ \Delta Q_N \end{bmatrix}$$

where $\Delta\delta_k = \delta_k^{v+1} - \delta_k^{v}$ and $\Delta V_k = V_k^{v+1} - V_k^{v}$ when v is the iteration index and

$$\begin{bmatrix} J_1 & J_2 \\ J_3 & J_4 \end{bmatrix} = \begin{bmatrix} \frac{\partial \Delta P}{\partial \delta} & \frac{\partial \Delta P}{\partial V} \\ \frac{\partial \Delta Q}{\partial \delta} & \frac{\partial \Delta Q}{\partial V} \end{bmatrix}$$

Newton-Raphson load flow method may have the final formulation of (Grisby 2007)

$$\begin{bmatrix}\Delta P\\ \Delta Q\end{bmatrix}=\begin{bmatrix}J_1 & J_2\\ J_3 & J_4\end{bmatrix}\begin{bmatrix}\Delta\delta\\ \Delta V\end{bmatrix}$$

Using Newton-Raphson method to solve the mismatch powers equations either in polar coordinates with **(**Δ**V**) and **(**Δ**δ)** as variables or in rectangular coordinates with **(**Δ**e**) and **(**Δ**f)** as variables [Al-Shakarchi 1973].

*** FAST DECOUPLED LOAD FLOW METHOD**

Fast decoupled load flow method, possibly the most popular method used by utilities, is well known for its speed of solution, reduced memory, and reliable convergence (Nanda 1987). The algorithm is simpler, faster and more reliable than Newton's method and has lower storage requirements. The fast decoupled load flow method is based on Newton's load flow method with the modifications of neglecting the J_2 and J_3 Jacobian submatrices due to the weak coupling between "P-V" and "Q-δ" quantities in power transmission system. Together with other approximations, the fast decoupled load flow equations become:

$$\left[\frac{\mathbf{\Delta}P}{V}\right]=\left[B'\right]\left[\mathbf{\Delta}\delta\right]$$

$$\left[\frac{\mathbf{\Delta}Q}{V}\right]=\left[B''\right]\left[\mathbf{\Delta}V\right]$$

where $B'_{km}=-\frac{1}{x_{km}}$ for m≠k and $B'_{kk}=\sum_{m\omega k}\frac{1}{x_{km}}$ for m=k

$B''_{km}=-B_{km}$ for m≠k and $B''_{kk}=\sum_{m\omega k}B_{km}$ for m=k (Stott and Alsac 1974)

GENETIC ALGORITHM

Genetic algorithms (GAs) are adaptive methods which may be used to solve search and optimization problems. Over many generations, natural populations evolve according to the principles of natural selection and "survival of the fittest". By mimicking this process, genetic algorithms are able to "evolve" solutions to real world problems, if they have been suitably encoded [Holland 1975]. Genetic algorithms work with a "population of individuals", each representing a possible solution to a given problem. Each individual is assigned a "fitness score" according to how good a solution to the problem it is. The highly-fit individuals are given opportunities to "reproduce", by "cross breeding" with other individuals in the population. This produces new individuals as "offspring", which share some features taken from each "parent". The least fit members of the population are less likely to get selected for reproduction, and so "die out". A whole new population of possible solutions is thus produced by selecting the best individuals from the current "generation", and mating them to produce a new set of individuals. This new generation contains a higher proportion of the characteristics possessed by the good members of the previous generation. In this way, over many generations, good characteristics are spread throughout the population. By favouring the mating of the more fit individuals, the most promising areas of the search space are explored. If the genetic algorithm has been designed well, the population will converge to an optimal solution to the problem. There are some differences between genetic algorithms and traditional searching algorithms (such as numerical techniques). They could be summarized as follows [Younes 2006]:

- The algorithms work with a population of strings, searching many peaks in parallel, as opposed to a single point.
- Genetic algorithms work directly with strings of characters representing the parameters set, not the parameters themselves.
- Genetic algorithms use probabilistic transition rules instead of deterministic rules.
- Genetic algorithms use objective function information instead of derivatives or other auxiliary knowledge (convexity, modality, continuity, differentiability).
- Genetic algorithms have the potential to find solutions in many different areas of the search space simultaneously.

GENETIC ALGORITHM IMPLEMENTATION

A simple genetic algorithm is an iterative procedure, which maintains a constant size population of candidate solutions. During each iteration step (generation), three genetic operators (reproduction, crossover, and mutation) are performing to generate new populations (offspring), and the chromosomes of the new populations are evaluated via the value of the fitness which is related to cost function. Based on these genetic operators and the evaluations, the better new populations of candidate solutions are performed [Younes 2006]. With the above description, the three steps in executing the genetic algorithm operating on fixed-length character strings are as follows:
1. Randomly create an initial population of individual fixed-length character strings.
2. Iteratively perform the following sub steps on the population of strings until the termination criterion has been satisfied:
A. Assign a fitness value to each individual in the population using the fitness measure.
B. Create a new population of strings by applying the following three genetic operations. The genetic operations are applied to individual string(s) in the population chosen with a probability based on fitness.
i. Reproduce an existing individual string by copying it into the new population.
ii. Create two new strings from two existing strings by genetically recombining substrings using the crossover operation at a randomly chosen crossover point.
iii. Create a new string from an existing string by randomly mutating the character at one randomly chosen position in the string.
3. The string that is identified by the method of result designation (e.g. the best-so-far individual) is designated as the result of the genetic algorithm for the run. This result may represent a solution (or an approximate solution) to the problem.

Now, we'll discuss briefly each step of the implementation of the genetic algorithm:

- CHROMOSOME REPRESENTATION

Genetic algorithms operate on representations of solutions to problems. Since they work with encoded parameters of the optimization problem, the choice of a representation form has a large impact on the performance. There are different ways of encoding solutions, and probably no single best way for all problems. The performance of genetic algorithms depends on the choice of a suitable representation technique. Most genetic algorithms applications use Holland’s fixed-length simple binary coding. This is historically the most widely used representation. Each chromosome is comprised of zeroes and ones, with each bit representing a gene [Abdul-Haleem 2005]. A conceptually simpler technique would be the real-coded representation, in which each chromosome vector is coded as a vector of floating point numbers of the same length as the solution vector. Each element was forced to be within the desired range, and the genetic operators were carefully designed to preserve this requirement [Michalewicz 1996].

- POPULATION INITIALIZATION

In the genetic algorithm, populations of chromosomes are created randomly by generating the required number of individuals using a random number generator that uniformly distributes numbers in the desired range. The extended random initialization is a variation whereby a number of random initializations are tried for each individual and the one with the best performance is chosen for the initial population. Other users of genetic algorithms have seeded the initial population with some individuals that are known to be in the vicinity of the global optimum. This approach is only applicable if the nature of the problem is well understood beforehand or if the genetic algorithm is used in conjunction with knowledge based system [Abdul-Haleem 2005].

- OBJECTIVE FUNCTION OR FITNESS FUNCTION

The objective function is used to provide a measure of how individuals have performed in the problem domain. In the case of a minimization problem, the mostly fit individuals will have the lowest numerical value of the associated objective function. This raw measure of fitness is usually only used as an intermediate stage in determining the relative performance of individuals in a genetic algorithm. Another function is the fitness function, is normally used to transform the objective function value into a measure of relative fitness [Ibrahim 2005].

- REPRODUCTION

The selection, or competition, is a stochastic process in which the chance of an individual surviving is proportional to its adaptation level. The adaptation is measured by the phenotype (search point, solution) evolution, that is, the characteristics presented by an individual in the problem environment (search space). The genetic algorithm, through selection, determines which individuals will go to the reproduction phase. There are several selection methods, where the fittest individuals from each generation are preferentially chosen for reproduction [Zamanan 2006]. Some of these methods are:
a. ROULETTE WHEEL SELECTION METHOD.
b. TOURNAMENT SELECTION METHOD.

- RECOMBINATION

Recombination produces new individuals in combining the information contained in two or more parents (parents-mating population). This is done by combining the variable values of the parents. Depending on the representation of the variables, different methods must be used. For the recombination of binary valued variables, the name “crossover” is established. This has mainly historical reasons. During the recombination of binary variables, only parts of the individuals are exchanged between the individuals. Depending on the number of parts, the individuals are divided before the exchange of variables (the number of cross points). The number of cross points distinguishes the methods.
In single-point crossover, one crossover position $\mathbf{a \in [\ 1, 2\ , \ldots\ldots\ , N_{var-1}\ ]}$, where ($\mathbf{N_{var}}$) is the number of variables of an individual, is selected uniformly at random and the variables exchanged between the individuals about this point, then two new offspring are produced.
In double-point crossover, two crossover positions are selected uniformly at random and the variables exchanged between the individuals between these points. Then two new offspring are produced. For multi-point crossover, (**c**) crossover positions $\mathbf{a_i \in [\ 1\ , 2\ , \ldots\ldots\ , N_{var-1}\ ];\ i = 1{:}\ m}$, where ($\mathbf{N_{var}}$) is the number of variables of an individual, are chosen at random with no duplicates and sorted into ascending order. Then, the variables between successive crossover points are exchanged between the two points to produce two new offspring. The section between the first variable and the first crossover point is not changed between individuals. Uniform crossover generalizes this scheme

to make every locus a potential crossover point. A crossover mask, the same length as the individual structure is created at random and the parity of the bits in the mask indicate which parent will supply the offspring with which bits [Pohlheim 2005].

- MUTATION

By mutation, individuals are randomly altered. These variations (mutation steps) are mostly small. They will be applied to the variables of the individuals with a low probability (mutation probability or mutation rate). Normally, offspring are mutated after being created by recombination [Pohlheim 2005]. In this process, randomly selected bits of randomly selected strings are changed from (**0**) to (**1**) and vice versa. This process occurs according to pre-specified probability. Usually, less than 5% of bits are changed in this process. Mutation process is used to escape from probable local optimum [Zamanan 2006].

- TERMINATION OF THE GENETIC ALGORITHM

Termination criteria or convergence criteria for genetic process may be triggered by finding an acceptable approximate solution and bring the search to halt. The termination criteria can be one or more of the following criteria [Abdul-Haleem 2005]:

a. Using diversity measure.
b. After a specified number of generations.
c. Finding an acceptable approximate solution.
d. Repetition until no change in the solution.

LOAD FLOW SOLUTION USING GENETIC ALGORITHM

The binary genetic algorithm is conceived to solve many optimization problems that stump traditional techniques. But, what if we are attempting to solve a problem where the values of the variables are continuous and we want to define them to the full machine precision? In such a problem, each variable requires many bits to represent it. If the number of variables is large, the size of the chromosome is also large. Of course, ones and zeros are not the only way to represent a variable. One could, in principle, use any representation conceivable for encoding the variables. When the variables are naturally quantized, the binary genetic algorithm fits nicely. However, when the variables are continuous, it is more logical to represent them by floating-point numbers. In addition, since the binary genetic algorithm has its precision limited by the binary representation of variables, using floating-point numbers instead easily allows representation to the machine precision. This continuous genetic algorithm also has the advantage of requiring less storage than the binary genetic algorithm because a single floating-point number represents the variable instead of ($\mathbf{N_{bits}}$) integers. The continuous genetic algorithm is inherently faster than the binary genetic algorithm, because the chromosomes do not have to be decoded prior to the evaluation of the cost function (objective function) [Haupt 2004].

COMPONENTS OF A CONTINUOUS GENETIC ALGORITHM

- THE VARIABLES AND COST FUNCTION

A cost function generates an output from a set of input variables (a chromosome). The cost function may be a mathematical function or an experiment. The objective is to modify the output in some desirable fashion by finding the appropriate values for the input variables. The goal is to solve some optimization problem where we search for an optimum (minimum) solution in terms of the variables of the problem. If the chromosome has ($\mathbf{N_{var}}$) variables (an N-dimensional optimization problem) given by ($\mathbf{b_1, b_2, \ldots\ldots\ldots, b_{Nvar}}$), then the chromosome is written as an array with ($\mathbf{1\times N_{var}}$) elements so that:

chromosome = [b_1, b_2, b_3, ………, b_{Nvar}]

In this case, the variable values are represented as floating-point numbers. Each chromosome has a cost found by evaluating the cost function (**f**) at the variables (**b_1, b_2, ………, b_{Nvar}**).

cost = f (chromosome) = f (b_1, b_2, ………, b_{Nvar})

Equations (10) and (11) along with applicable constraints constitute the problem to be solved [Haupt 2004].
Our primary problem in this research is the continuous functions introduced below. The two cost functions are [Kubba 1987]:

$$\Delta P_k = P_k^{sp} - V_k \sum_{m=1}^{n} V_m \left(G_{km} \cos\delta_{km} + B_{km} \sin\delta_{km}\right)$$

for "**PV**" and "**PQ**" busses

$$\Delta Q_k = Q_k^{sp} - V_k \sum_{m=1}^{n} V_m \left(G_{km} \sin\delta_{km} - B_{km} \cos\delta_{km}\right)$$

for "**PQ**" busses only

Where $\delta_{km} = \delta_k - \delta_m$, and
(**ΔP_k**) is the mismatch active power at bus (**k**) and (**ΔQ_k**) is the mismatch reactive power at bus (**k**). (**V_k, V_m**, δ_k, δ_m) are the voltage magnitude and angle at busses (**k**) and (**m**) respectively, which are the variables of the two cost functions.

*** VARIABLE ENCODIND, PRECISION, AND BOUNDS**
Here, the difference between binary and continuous genetic algorithms is shown. It is no longer needed to consider how many bits are necessary to represent accurately a value. Instead, (**V**) and (δ) have continuous values that are limited between appropriate bounds (which are in our problem, **$0.95 \le V \le 1.05$** and **$-5^\circ \le \delta \le 5^\circ$** for "**5 busbars**" typical test system and $0.9 \le V \le 1.1$, $-20^o \le \delta \le 20^o$ for Iraqi National Grid). Although the values are continuous, a digital computer represents numbers by a finite number of bits. When we refer to the continuous genetic algorithm, it means that the computer uses its internal precision and roundoff to define the precision of the value. Now, the algorithm is limited in precision to the roundoff error of the computer [Haupt 2004].

*** INITIAL POPULATION**
The genetic algorithm starts with a group of chromosomes known as the population. We define an initial population of (**N_{ind}**) chromosomes. A matrix represents the population with each row in the matrix being a (**$1 \times N_{var}$**) array (chromosome) of continuous values. Given an initial population of (**N_{ind}**) chromosomes, the full matrix of (**$N_{ind} \times N_{var}$**) random values is generated.
All variables are normalized to have values between (**0**) and (**1**), the range of a uniform random number generator. The values of a variable are "unnormalized" in the cost function. If the range of values is between (**b_{lo}**) and (**b_{hi}**), then the unnormalized values are given by:

$$b = (b_{hi} - b_{lo})\, b_{norm} + b_{lo}$$

where:

$\mathbf{b_{hi}}$: highest number in the variable range.
$\mathbf{b_{lo}}$: lowest number in the variable range.
$\mathbf{b_{norm}}$: normalized value of variable.
This society of chromosomes is not a democracy: the individual chromosomes are not all created equal. Each one's worth is assessed by the cost function. So at this point, the chromosomes are passed to the cost function for evaluation [Haupt 2004].

- NATURAL SELECTION
Survival of the fittest translates into discarding the chromosomes with the highest cost. First, the ($\mathbf{N_{ind}}$) costs and associated chromosomes are ranked from lowest cost to highest cost. Then, only the best are selected to continue, while the rest are deleted. The selection rate, ($\mathbf{X_{rate}}$), is the fraction of ($\mathbf{N_{ind}}$) that survives for the next step of mating. The number of chromosomes that are kept each generation is:

$$\mathbf{N_{keep} = X_{rate} \cdot N_{ind}}$$

Natural selection occurs each generation or iteration of the algorithm. Of the ($\mathbf{N_{ind}}$) chromosomes, only the top ($\mathbf{N_{keep}}$) survive for mating, and the bottom ($\mathbf{N_{ind} - N_{keep}}$) are discarded to make room for the new offspring. Deciding how many chromosomes to keep is somewhat arbitrary. Letting only a few chromosomes survive to the next generation limits the available genes in the offspring. Keeping too many chromosomes allows bad performers a chance to contribute their traits to the next generation. We often keep 50% ($\mathbf{X_{rate}=0.5}$) in the natural selection process [Haupt 2004].

- SELECTION FOR MATING
In this process, two chromosomes are selected from the mating pool of ($\mathbf{N_{keep}}$) chromosomes to produce two new offspring. Pairing takes place in the mating population until ($\mathbf{N_{ind} - N_{keep}}$) offspring are born to replace the discarded chromosomes. Pairing chromosomes in a genetic algorithm can be as interesting and varied as pairing in an animal species. We'll look at a variety of selection methods.
a. Weighted random pairing (roulette-wheel): which is divided into:
i. Rank weighting.
ii. Cost weighting.
b. Tournament selection.
Each of the parent selection schemes results in a different set of parents. As such, the composition of the next generation is different for each selection scheme. Roulette-wheel and tournament selection are standard for most genetic algorithms. It is very difficult to give advice on which selection scheme works best. In our problem, we follow the rank-weighting roulette-wheel and tournament parent selection procedures [Haupt 2004].

- RECOMBINATION
As for the binary algorithm, two parents are chosen, and the offspring are some combination of these parents. Many different approaches have been tried for crossing over in continuous genetic algorithm. The simplest methods choose one or more points in the chromosome to mark as the crossover points. Then the variables between these points are merely swapped between the two parents. The problem with real-valued crossover methods is that no new information is introduced: each continuous value that was randomly initiated in the initial population is propagated to the next generation, only in different combinations. Although this strategy works fine for binary representations, there is now a continuum of values, and in this continuum we are merely interchanging two data points. These approaches totally rely on mutation to introduce new genetic

material. The blending models remedy this problem by finding ways to combine variable values from the two parents into new variable values in the offspring.
In our problem, we want to find a way to closely mimic the advantages of the binary genetic algorithm scheme. It begins by randomly selecting a variable in the first pair of parents to be the crossover point:

$$\mathbf{c = roundup\ \{random * N_{var}\}}$$

We'll let

$$\text{parent } 1 = [b_{m1}, b_{m2}, \ldots\ldots, b_{mc}, \ldots\ldots, b_{mNvar}]$$
$$\text{parent } 2 = [b_{d1}, b_{d2}, \ldots\ldots, b_{dc}, \ldots\ldots, b_{dNvar}]$$

Where (**m**) and (**d**) subscripts discriminate between the mom and dad parent. Then, the selected variables are combined to form new variables that will appear in the children:

$$b_{new1} = b_{mc} - \beta\,(b_{mc} - b_{dc})$$
$$b_{new2} = b_{dc} + \beta\,(b_{mc} - b_{dc})$$

Where **β** is a random value between **0** and **1**. The final step is to complete the crossover with the rest of the chromosome as in binary genetic algorithm:

$$\text{offspring } 1 = [b_{m1}, b_{m2}, \ldots\ldots, b_{new1}, \ldots\ldots, b_{dNvar}]$$
$$\text{offspring } 2 = [b_{d1}, b_{d2}, \ldots\ldots, b_{new2}, \ldots\ldots, b_{mNvar}]$$

If the first variable of the chromosomes is selected, then only the variables to the right of the selected variable are swapped. If the last variable of the chromosomes is selected, then only the variables to the left of the selected variable are swapped. This method does not allow offspring variables outside the bounds set by the parent unless **β > 1** [Haupt 2004].

* MUTATION

Random mutations alter a certain percentage of the genes in the list of chromosomes. We can sometimes find our method working too well. If care is not taken, the genetic algorithm can converge too quickly into one region of the cost surface. If this area is in the region of the global minimum, that is good. However, some functions, such as the one we are modeling, have many local minima. If nothing is done to solve this tendency to converge quickly, it may end up in a local rather than a global minimum. To avoid this problem of overly fast convergence (premature convergence), the routine is forced to explore other areas of the cost surface by randomly introducing changes, or mutations, in some of the variables. The basic method of mutation is not much more complicated for the continuous genetic algorithm. Do we also allow mutations on the best solutions? Generally not. They are designated as elite solutions destined to propagate unchanged. Such elitism is very common in genetic algorithms. Why throw away a perfectly good answer? The equation of mutation used here is [Haupt 2004]:

$b_{mut} = (b_{hi} - b_{lo})\, b_{norm} + b_{lo}$; b_{mut}: variable under mutation.

MULTIPLE OBJECTIVE OPTIMIZATION (MOO)

In many applications, the cost function has multiple, often times, conflicting objectives. The most important approach to (MOO) are: **weighted cost functions** [Haupt 2004].

$$\mathbf{cost} = \sum_{i=1}^{N} w_i f_i$$

Where:

$\mathbf{f_i}$ is the cost function (**i**).

$\mathbf{w_i}$ is the weighting factor and $\sum_{i=1}^{N} w_i = 1$.

The problem with this method is determining appropriate values of ($\mathbf{w_i}$). Different weights produce different costs for the same ($\mathbf{f_i}$). This approach requires assumptions on the relative worth of the cost functions prior to running the genetic algorithm. Implementing this multiple objective optimization approach in a genetic algorithm only requires modifying the cost function to fit the form of eq. (17) and does not require any modification to the genetic algorithm [Haupt 2004]. Thus,

$$\mathbf{cost} = w f_1 + (1-w) f_2$$

This approach is adopted in this research for its simplicity, easy of programming and gives us the required accuracy. Here, (**w**) is chosen to be (**0.5**).

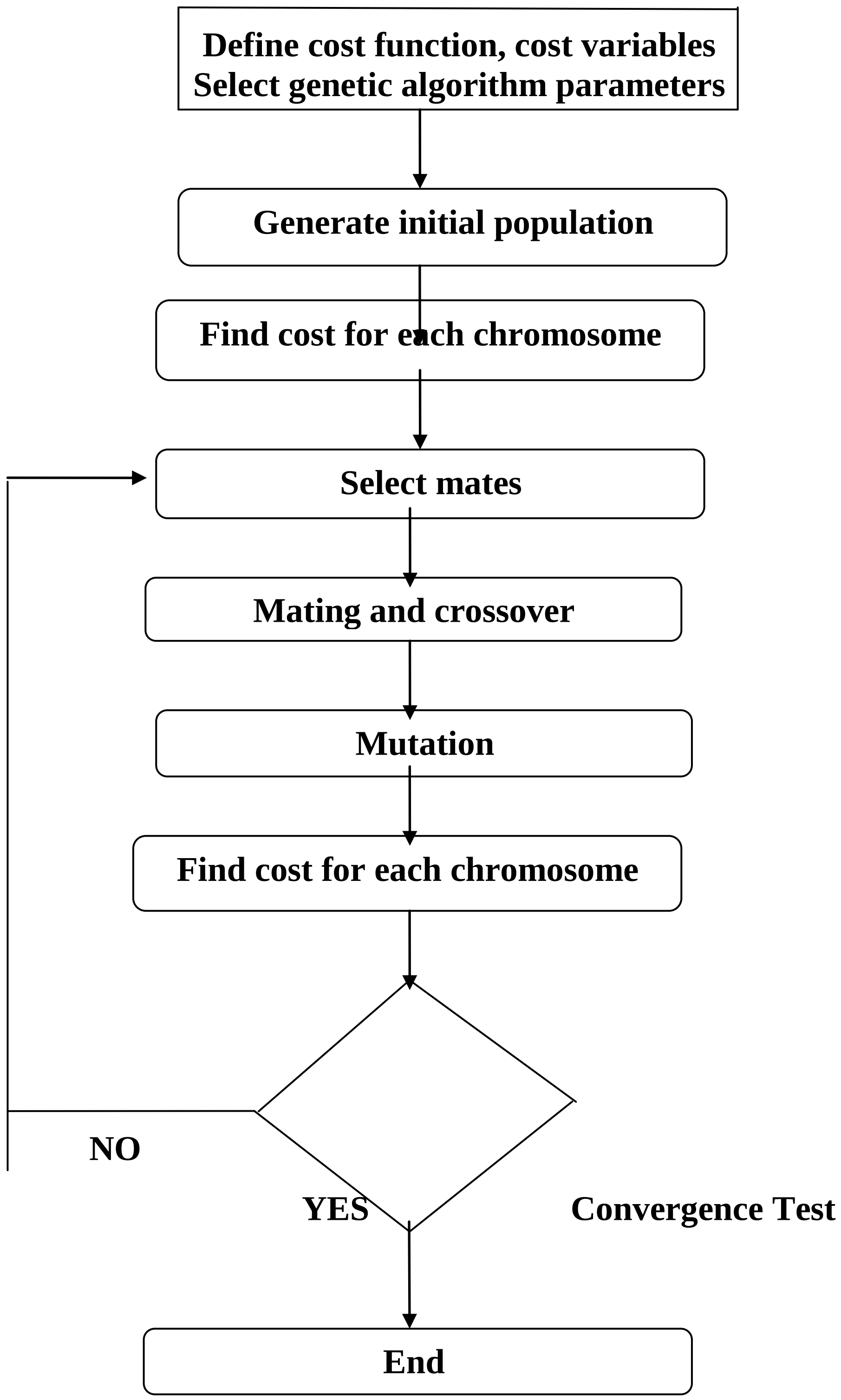

Define cost function, cost variables
Select genetic algorithm parameters
Generate initial population
Find cost for each chromosome
Select mates
Mating and crossover
Mutation
Find cost for each chromosome
NO
YES
Convergence Test
End

Fig. (2) Flowchart of a Continuous Genetic Algorithm

IMPLEMENTATION AND RESULTS

The proposed continuous (real-coded) genetic algorithm is demonstrated on two test systems namely, 5-buses test system [Stagg 1968] which lines data and buses data are present in appendix. The second system is the 362-bus Iraqi National Grid (ING) with 599 branches. The software is implemented by using MATLAB version 7 with Pentium 4, 2 GHz CPU, and 2 Gbyte RAM. The input data are the nodal admittance matrix, buses data (specified values), and the slack voltage

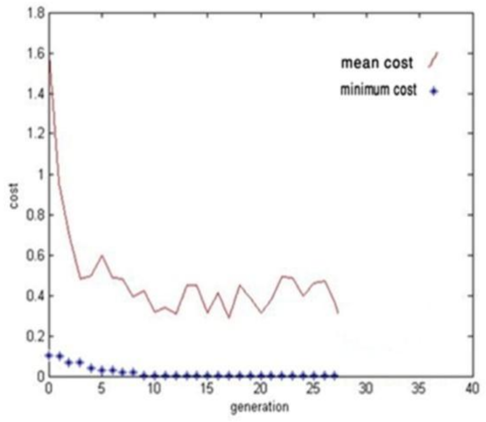
1. Five Buses Typical Test System Results

Table (1) shows the results of the application of the genetic algorithm to "**5 bus-bars**" typical test system to find the active and reactive power mismatches described in eqs. (12) and (13) for accuracy of **0.001 p.u** (**0.1 MW/MVAR)** and the voltage magnitude and phase angle associated with each busbar. Because of the stochastic nature of the genetic algorithm process, each independent run will probably produce a different number of generations and consequently the computation time and the best amongst these should be chosen. The best of the (**10**) implementation runs are shown in this table. The base quantities are 132 KV and 100 MVA.

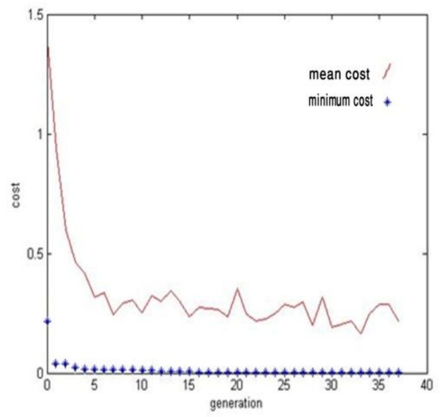
Table (2) Load Flow Solution for "5 busbars" Typical Test System with an Accuracy of (0.001)

Bus No.	Active power mismatch(p.u)	Reactive power mismatch(p.u)	Voltage magnitude(p.u)	Voltage angle(deg.)	No. of generations
1	slack	slack	1.06	0.00	—
2	0.00018139	0.00076184	1.0333	-0.2856	37
3	0.00094649	0.00028622	1.0349	-2.5267	27
4	0.00096724	0.0004207	1.0014	-1.4736	26
5	0.00091083	0.00096267	0.9786	-1.5042	35
Total	**Computational**	**Time:**			**0.516 sec.**

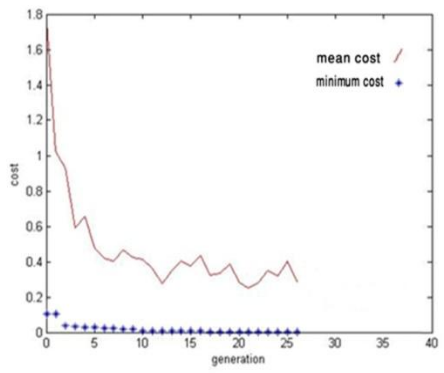
Figures below, the dotted curves show the evolution process of the active and reactive power mismatches (cost function) versus the number of generations and the solid curves show the mean cost of the individuals versus the number of generations at each busbar of the "**5 busbars**" typical test system with an accuracy of (**0.001**). In the figures and due to the stochastic nature of the genetic algorithm, we note that each busbar requires different number of generations to reach the required accuracy (0.001 p.u.).

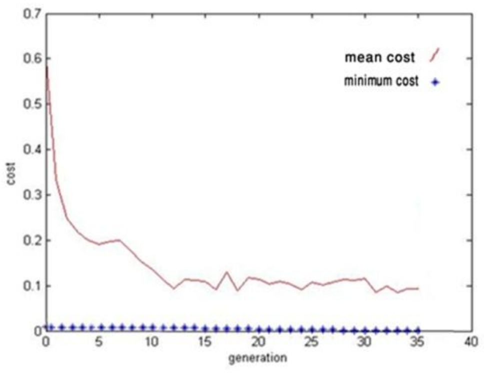


Evolution Process at Busbar 2



Evolution Process at Busbar 3





Evolution Process at Busbar 4 **Evolution Process at Busbar 5**

In the table (2) shown, the active and reactive line flows (power flows) for the "**5 busbars**" typical test system which consists of (**7**) lines are calculated, the calculation is done for an accuracy of the active and reactive power mismatches of (**0.001**).

Table (2) Active and Reactive Power Flows for the "5 busbars" Typical Test System with an Accuracy of (0.001)

Line No.	Line Term	Active power flow (MW)	Reactive power flow (MVAR)
1	1—2	22.3473	-39.7436
2	1—3	21.5944	-4.3321
3	2—3	20.7693	7.3872
4	2—4	16.2575	-13.0168
5	2—5	17.1768	-63.7962
6	3—4	-22.2970	-123.5797
7	4—5	-3.0479	-21.5199

* Iraqi National Grid Load Flow Solution Results:

The 362-bus (ING) consists of 30-generator bus and 332-load bus with 599 branches [Al-Bakri 1994]. Table (3) illustrates the load flow solution for Iraqi National Grid using the proposed continuous (real-coded) Genetic Algorithm with a power tolerance (minimum active and reactive power mismatches) of 0.001 p.u. with base quantities of 132 KV and 100 MVA. Also, the number of iterations (generations) for each bus are tabulated. Due to the huge table, only the load flow solution results of 96 bus are tabulated in this paper.

Table (3) Load Flow Solution for "IRAQI NATIONAL GRID" with an Accuracy of (0.001)

Bus No.	Active power mismatch (p.u)	Reactive power mismatch (p.u)	Voltage magnitude (p.u)	Voltage angle (p.u)	No. of generations
1	slack	slack	1.04	0	—
2	0.0005	PV	1	18.3805	262
3	0.00021334	PV	1	2.8233	57
4	0.0008081	PV	1	-9.5400	3191
5	0.00011245	PV	1	13.6445	521
6	0.00043106	PV	1	-11.8520	34
7	0.0018487	PV	1	4.1875	5000
8	0.00066843	PV	1	7.5529	244
9	0.00023882	PV	1	12.3150	30
10	0.00016648	PV	1	4.0006	134
11	0.0003391	PV	1	-19.7704	884
12	0.00045458	PV	1	-6.3530	266
13	0.00013682	PV	1	4.5221	424
14	0.00058912	PV	1	-6.9794	76
15	0.00054176	PV	1	-8.1968	353
16	0.00021063	PV	1	13.5898	42
17	0.00078201	PV	1	4.5766	39
18	$2.4477*10^{-6}$	PV	1	11.1094	415
19	0.00090163	PV	1	7.0672	47
20	0.00089409	PV	1	-7.0275	9
21	0.00037127	PV	1	-3.2876	159
22	0.00014522	PV	1	-10.7986	24
23	0.00093387	PV	1	2.0421	216
24	0.00084462	PV	1	9.0268	472
25	0.00038532	PV	1	2.9669	17
26	0.00023586	PV	1	3.8338	527
27	$7.2047*10^{-6}$	PV	1	-6.8666	88
28	0.00011686	PV	1	0.0252	50
29	0.00026843	PV	1	7.3612	333
30	0.0005791	PV	1	9.1833	1342

Bus No.	Active power mismatch (p.u)	Reactive power mismatch (p.u)	Voltage magnitude (p.u)	Voltage angle (p.u)	No. of generations
31	0.0003	0.0002	1.02247	12.3782	36
32	0.00031048	0.00013264	1.03356	-12.5347	57
33	0.00076127	0.00088425	0.915482	-18.8922	747
34	0.00048097	$7*10^{-5}$	0.981021	-3.14901	60
35	0.00082233	0.00019414	0.909731	-7.5778	927
36	0.0007	0.0004	0.999866	0.40992	54
37	0.0009483	0.00059979	0.99142	-0.108361	3
38	0.00072258	$2.708*10^{-5}$	1.02766	2.41241	393
39	0.00010822	$3.1427*10^{-5}$	0.911085	8.90456	21
40	$7.3767*10^{-5}$	$8.2306*10^{-5}$	0.978972	7.68687	21
41	$3.9217*10^{-5}$	0.00014068	0.914069	-2.09315	30
42	0.00056231	0.00034232	0.931974	10.2677	117
43	0.00015186	0.00084486	0.916778	9.06423	69
44	0.00093275	0.00060546	0.935108	-13.2829	18
45	0.00085845	$3.9338*10^{-5}$	1.00216	-8.97243	12
46	$2.0578*10^{-5}$	0.00022034	0.935224	-7.08481	54
47	0.0005849	0.00025364	0.96548	-8.2154	120
48	0.0006	0.0009	0.96959	0.0359168	9
49	0.00046407	0.00068482	1.01733	8.11071	36
50	0.00064014	0.00040881	0.906882	16.9887	3
51	0.00081139	0.00073374	1.01725	12.1607	15
52	0.00063582	0.00036833	0.999238	0.015595	12
53	0.00064585	0.00073222	0.924709	7.0377	54
54	0.00039461	0.00031144	1.02049	4.97693	54
55	0.00050197	0.00051873	0.912678	18.5084	312
56	0.0006757	0.00066047	1.02961	13.4216	57
57	0.00048624	0.00046999	1.01989	13.0202	165
58	0.00026657	0.00078447	1.03154	-4.56727	54
59	0.00020212	0.00087574	0.998382	-0.133244	135
60	0.00099704	0.00040061	0.969316	-9.13924	336
61	0.0002	0.0005	1.09633	-3.3891	60

62	0.00062866	0.00048481	1.01166	3.0274	69
63	$3.4536*10^{-6}$	0.00050729	1.00118	11.0695	144

Bus No.	Active power mismatch (p.u)	Reactive power mismatch (p.u)	Voltage magnitude (p.u)	Voltage angle (p.u)	No. of generations
64	0.00052304	$6.1774*10^{-5}$	0.926323	0.126932	30
65	0.00039148	0.00027503	0.959269	-3.93418	234
66	0.0003338	0.00048038	0.939048	5.45821	24
67	0.00045369	0.00074015	1.03691	8.05995	12
68	0.00077681	0.00053799	0.962027	-0.556099	186
69	0.00027057	0.00061177	1.0712	5.11859	144
70	0.00040663	0.00054506	0.909477	-13.4194	18
71	0.00015602	0.00084029	1.0151	4.6136	60
72	0.00085314	$4.4098*10^{-5}$	0.9505	-9.4481	54
73	0.00082757	0.00056498	0.9587	14.8084	132
74	$6.6466*10^{-5}$	0.00028761	1.0205	-0.5232	24
75	0.00045213	0.00089712	0.9106	16.254	450
76	0.00026312	0.00099152	1.0235	-0.4562	60
77	0.0003	0.0002	0.9989	-15.9269	1137
78	0.00060173	0.00081968	0.9069	9.3835	102
79	0.00097288	0.00093161	0.9115	-9.6373	399
80	0.00030775	0.00026214	1.0172	-11.3539	567
81	0.00028197	0.0003666	0.9567	10.958	72
82	0.0001	0.0007	0.9009	-9.6128	1215
83	0.00017501	0.00031678	0.9538	0.5662	36
84	0.00060463	0.00055344	0.9389	12.4227	51
85	0.00049479	0.00025939	0.9917	-17.9393	69
86	0.00074739	$6.1917*10^{-5}$	0.9976	11.2733	54
87	0.00073692	0.00062359	1.0459	3.6541	90
88	0.0002	0	0.9265	7.1323	69
89	$8.0049*10^{-5}$	0.00089968	1.0154	-0.9266	33
90	0.00051316	0.00093613	1.0007	-15.3443	189
91	$3.1397*10^{-5}$	0.00072166	1.0017	18.7953	138
92	0.00062835	0.00047428	0.9977	12.0889	27

93	$4.7755*10^{-5}$	0.00047299	1.0034	11.0409	60
94	0.00031952	0.00085516	0.9855	1.9537	135
95	0.00028333	0.00050642	1.0368	18.9362	60
96	0.0008504	0.00096063	1.0821	-15.1242	96

- CONCLUSION

The proposed method (Real-Coded Genetic Algorithm) presented in this paper can be implemented on-line for small and medium-scale power systems load flow solution and it can be used for planning study for large-scale systems. The proposed method has reliable convergence and high accuracy of solution. Whereas the traditional numerical techniques (Gauss-Seidel, Newton-Raphson, Fast decoupled,…etc.) use the characteristics of the problem to determine the next sampling point (e.g. gradient, linearity and continuity), genetic algorithm makes no such assumptions. Instead, the next sampled point is determined based on stochastic sampling or decision rules rather than on a set of deterministic decision rules. Also, whereas the traditional numerical techniques mentioned above use a single point at a time to search the problem space, genetic algorithm uses a population of candidate solutions for solving the problem. Thus, reducing the possibility of ending at a local minima. Although binary-coded genetic algorithm has been successfully applied to a wide range of optimization problems, they suffer from disadvantage when applied to the real-world problems involving a large number of variables. Thus, we use in our problem the real-coded genetic algorithm, where all decision variables (unknowns) are expressed as real numbers. Explicit conversion to binary does not take place. A reduction of computational effort is an obvious advantage of a real-coded genetic algorithm. Another advantage is that, an absolute precision is now attainable by making it possible to overcome the crucial decision of how many bits are needed to represent potential solutions. Blending models are used in the crossover operator to remedy the problem of the crossover in the real-coded genetic algorithm which is, no new information is introduced: each continuous value that was randomly initiated in the initial population is propagated to the next generation, only in different combinations. Thus, the blending methods combine variable values from the two parents into new variable values in the offspring. At the same time, these methods do not allow offspring variables outside the bounds set by the parent unless $\boldsymbol{\beta} > \mathbf{1}$, where $\boldsymbol{\beta}$ is a random number on the interval $[\mathbf{0},\mathbf{1}]$. Finally, solving the load flow problems by genetic algorithm gives high accurate results with respect to the conventional methods, since load flow study is multiple solutions.

REFRENCES

- Abdul-Haleem G. F., 2005, “A Genetic Algorithm for Manufacturing Cell Formation”, M.Sc Thesis, Mechanical Department, University of Baghdad.
- Al-Shakarchi M. R. G., 1973, “Nodal Iterative Load Flow”, A dissertation submitted to the Victoria University of Manchester.
- AL-BAKRI A. A., 1994, “A Study of Some Problems on the Iraqi National Grid and Establishing a Method Algorithm for Load Flow”, M.Sc. Thesis, University of Baghdad.
- Grisby Leonard L., 2007, “Power Systems”, CRC Press.
- Haupt R. L. and Haupt S. E., 2004, “Practical Genetic Algorithms”, A John Wiley & Sons, INC., Publication, 2nd edition.
- Holland J., 1975, “Adaptation in Natural and Artificial Systems”, MIT Press.
- Ibrahim S. B. M., 2005, “The PID Controller Design Using Genetic Algorithm”, A dissertation submitted to University of Southern Queensland, Faculty of engineering and surveying, Electrical and Electronics Engineering.
- Kubba H.A., 1987, “Comparative Study of Different Load Flow Solution Methods”, M.Sc Thesis, University of Baghdad.
- Michalewicz Z., 1996, “Genetic Algorithms + Data Structure = Evolution Programs”, AI series, Springer-Verlag, New York, 3rd edition.
- Nanda J., Kothari D. P. and Srivastava S. C., 1987, “Some Important Observations on Fast Decoupled Load Flow Algorithm”, Proceedings of the IEEE, VOL. 75, No. 5, 732-733
- Pohlheim H., 2005, “GEATBx: Genetic and Evolutionary Algorithm Toolbox for Use with Matlab”, available at http://www.geatbx.com/.
- Stott B. and Alsac O., 1974, “Fast Decoupled Load Flow”, IEEE Trans. Power App. Syst., VOL. PAS-93, 859-869
- STAGG G. W. and AL-ABIAD A., 1968, “Computer Methods in Power System Analysis”, Mc-Graw Hill Publishing Company.
- Talib A., 2007, “An Optimization Approach of Robot Motion Planning Using Genetic Algorithm”, M.Sc Thesis, Mechatronics Department, AL-Khwarizmi Engineering, University of Baghdad.

- Taylor D. G. and Treece J. A., 1967, "Load Flow Analysis by Gauss-Seidel Method", presented at the Symp. on power systems load flow analysis, University of Manchester Institute of Science and Technology, Manchester, U.K.

- Woon L. C., 2004, "Genetic Algorithm for Load Flow Solution Techniques", Abstract of thesis, Master of engineering (electrical), http://www.sps.utm.

- Younes M. and Rahli M., 2006, "On the Choice Genetic Parameters with Taguchi Method Applied in Economic Power Dispatch", Leonardo journal of sciences, issue 9, pp. 9-24.

- Zamanan N., Sykulski J., Al-Othman A. K., 2006, "A Digital Technique for Online Identification and Tracking of Power System Harmonics Based on Real Coded Genetic Algorithm", Proceedings of the sixth IASTED international conference, European power and energy systems, Rhodes, Greece.

LIST OF SYMBOLS AND ABBREVIATIONS

b : Any variable in the chromosome.
B : Imaginary part of admittance.
c : Crossover point.
e : Real part of bus voltage.
E : Complex bus voltage.
f : Imaginary part of bus voltage.
G : Real part of admittance.
GA : Genetic Algorithm.
LF : Load Flow.
n : Number of busses in the power system.
N : Number of objective functions.
N_{var} : Number of variables in an individual.
N_{keep} : Number of chromosomes that are kept each generation.
N_{ind} : Number of individuals in the population.
p : Iteration index.
p.u : Per unit.
PQ : Load busbars.
PV : Generator busbars.
sp : Specified value.
V : Bus voltage magnitude.
w : Weighting factor.
X_{rate} : Selection rate.
Y_{kk} : Self admittance of bus (k).
Y_{km} : Branch admittance between busses (k) and (m).
ΔP : Active power mismatch.
ΔQ : Reactive power mismatch.
δ : Bus voltage phase angle.
β : Random number on the interval [0,1].

APPENDIX I

Nodal admittance matrix elements for “**5 busbars**” typical test system, per-unit quantity = **100 MVA, and 132 KV.** The following data are all in (**p.u**), and buses data.

From Bus	To Bus	G (p.u)	B (p.u)
1	1	6.25	-18.695
1	2	-5	15
1	3	-1.25	3.75
1	4	0	0
1	5	0	0
2	2	10.83334	-23.415
2	3	-1.66667	5
2	4	-1.66667	5
2	5	-2.5	7.5
3	3	12.91667	-38.695
3	4	-10	30
3	5	0	0
4	4	12.91667	-38.695
4	5	-1.25	3.75
5	5	3.75	-11.21

Bus No.	Specified active power (p.u)	Specified reactive power (p.u)	Voltage (p.u)
1	slack	slack	1.06+j0
2	0.2	0.2	—
3	-0.45	-0.15	—
4	-0.4	-0.05	—
5	-0.6	-0.1	—