A Multi-objective Real-Coded Genetic Algorithm for a Rapid and Efficient Load Flow Solution of Power Systems

[1]Asst. Prof. Dr. Hassan Kubba, [2]Moneer Thamir Esmael

University of Baghdad, College of Engineering
Department of Electrical Engineering
M.Sc. Electrical Engineering

Abstract-- **This paper presents a multi-objective real-coded Genetic Algorithm (RCGA) to solve the load flow problem (LFP). Since the load flow problem has multiple solutions, the local minima and the premature convergence are some of the drawbacks of the conventional methods. The GA is a search problem which uses a population of candidate solutions for solving the problem, thus reducing the possibility of ending at a local minima. A real-coded multi-objective GA is used since LFP involves a large number of variables, 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 GA. 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.**

In LFP the cost function has two conflicting objectives, which are the mismatch active and reactive powers. The most straightforward approach to multi-objective optimization is the "Sum of Weighted Cost Functions". This approach is to weight each function and add them together. This approach is adopted in this research for its simplicity, easy of programming and gives the required accuracy. The application of GA for solving LFP of small-scale systems is on-line (real-time) solutions. But for large-scale systems, the CPU computational time for high accuracy convergence is large. In this paper, a sparsity technique and optimal ordering for the sparse matrices (which have too many elements of zero values) are implemented in order to reduce the storage requirement and simplify some arithmetic operations to reduce the total computing time for high accurate solution. The sparsity technique is achieved by entering only the non-zero elements; also two identification vectors are needed to identify the exact location of the elements in the original matrix. The proposed method was demonstrated on different test systems, such as 14-bus and 30-bus IEEE test systems, and a 362-bus with 599-branches IRAQI NATIONAL GRID. From the obtained results, it is concluded that the proposed method presents a highly accurate solution for the unknown variables, a large saving in storage requirements and a reasonable reduction in total computing time.

Keywords: **Genetic Algorithm Operators, Load Flow Problem, Multi-Objectives Minimization, Real-Coded Genetic Algorithm, Sparsity Technique**

I. INTRODUCTION

The power flow problem, which is performed to determine the power system static states (voltage magnitudes and voltage phase angles) at each busbar to find the steady state operating condition of a system, is very important and the most frequently carried out study by electrical power utilities for power system on-line operation, planning and control. The mathematical formulation of the electrical power flow problem results in a set of non-linear algebraic equations. The power flow problem has multiple solutions [1]. The numerical methods and some of the artificial intelligence methods suffer from the local minima problem. Also there are many criteria which should be taken into consideration such as the speed of solution, storage requirement and the degree of solution accuracy. 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 a wide range of optimization problems. Because of existence of local optima, these algorithms offer promise in solving large-scale problems. An 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 [2]. 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.

II. THE REAL-CODED GENETIC ALGORITHM (RCGA)

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 real-coded 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 RCGA 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*) [2].

III. MATHEMATICAL DESCRIPTION AND RCGA OPERATORS

This genetic algorithm is very similar to the binary genetic algorithm, but the primary difference is the fact that variables are no longer represented by bits of zeros and ones, but instead by floating-point numbers over whatever range is deemed appropriate. However, this simple fact adds some nuances to the application technique that must be carefully considered. In particular, it will be presented different crossover and mutation operators.

A. 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 from 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. The term fitness is extensively used to designate the output of the *objective function* in the genetic algorithm literature. Fitness implies a maximization problem. Fitness has a closer association with biology than the term cost, thus genetic algorithm mimics *Darwin's evolution process* by we have adopted the term cost, since most of the optimization literature deals with minimization, hence cost. They are equivalent. If the chromosome has $\mathbf{N_{var}}$ variables (N-dimensional optimization problem) given by $(\mathbf{b_1, b_2, \ldots\ldots, b_{Nvar}})$, then the chromosome is written as an array with $(\mathbf{1\times N_{var}})$ elements so that:

$$\text{chromosome} = [b_1, b_2, b_3, \ldots\ldots\ldots, b_{Nvar}] \quad (1)$$

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 $(\mathbf{b_1, b_2, \ldots, b_{Nvar}})$.

$$\text{cost} = f(\text{chromosome}) = f(b_1, b_2, \ldots, b_{Nvar}) \quad (2)$$

Equations (1) and (2) along with applicable constraints constitute the problem to be solved. Our primary problem in this research is the continuous functions introduced below. The two cost functions are:

$$\Delta P_i = P_i^{sp} - \left| V_i \right| \sum_{k=1}^{N} \left| V_k \right| (G_{ik}\cos\theta_{ik} + B_{ik}\sin\theta_{ik}) \quad (3)$$

for generator buses, and load buses, "sp" is specified value

$$\Delta Q_i = Q_i^{sp} - \left| V_i \right| \sum_{k=1}^{N} \left| V_k \right| (G_{ik}\sin\theta_{ik} - B_{ik}\cos\theta_{ik}) \quad (4)$$

for load busses only, where $\boldsymbol{\theta_{ik} = \theta_i - \theta_k}$, and $\mathbf{y_{ik} = G_{ik} - jB_{ik}}$ is the branch admittance between buses i and k. The nodal admittance matrix **[Y]** of a power system, which is the main input data to the computerized algorithm, is a highly sparse, square and symmetric matrix. $\boldsymbol{\Delta P_i}$ is the mismatch active power at bus (i) and $\boldsymbol{\Delta Q_i}$ is the mismatch reactive power at bus (i). $(\mathbf{V_i}, \mathbf{V_k}, \boldsymbol{\theta_i}, \boldsymbol{\theta_k})$ are the voltage magnitude and angle at busses (i) and (k) respectively, which are the variables of the two cost functions and N is the number of buses [3].

B. Variable encoding, precision and bounds

Here, the difference between binary and RCGA is shown. It is no longer needed to consider how many bits are necessary to represent accurately a value. Instead, (**V**) and (**θ**) have continuous values that are limited between appropriate bounds which are in our problem, $\mathbf{0.95 \le |V| \le 1.05}$ p.u. and $\mathbf{-20 \le \theta \le 20}$ **[3]**. Although the values are continuous, a digital computer represents numbers by a finite number of bits. When we refer to the RCGA, it means that the computer uses its internal precision and round off to define the precision of the value. Now, the algorithm is limited in precision to the round off error of the computer. Since the genetic algorithm is a search technique, it must be limited to exploring a reasonable region of variable space. Sometimes, this is done by imposing a constraint on the problem. If one does not know the initial search region, there must be enough diversity in the initial population to explore a reasonably sized variable space before focusing on the most promising regions.

C. Initial population

The genetic algorithm starts with a group of chromosomes known as the *population*. We define an initial population of ($\mathbf{N_{ind}}$) chromosomes. A matrix represents the population with each row in the matrix being a ($\mathbf{1\times N_{var}}$) array (chromosome) of continuous values. Given an initial population of ($\mathbf{N_{ind}}$) chromosomes, the full matrix of ($\mathbf{N_{ind}\times N_{var}}$) 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 ($\mathbf{b_{lo}}$) and ($\mathbf{b_{hi}}$), then the unnormalized values are given by:

$$b=(b_{hi}-b_{lo})b_{norm}+b_{lo} \quad (5)$$

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. In this research, we use a population size of 20 chromosomes for 14-bus IEEE system, 200 chromosomes for 30-bus IEEE systems and 500 chromosomes for Iraqi National Grid.

D. 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:

$$N_{keep} = X_{rate}.N_{ind} \quad (6)$$

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. 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 use 50% ($\mathbf{X_{rate}=0.5}$) in the natural selection process. Another approach to natural selection is called *thresholding* is used in this research. In this approach, all chromosomes that have a cost lower than some threshold survive. An attractive feature of this technique is that the population does not have to be sorted.

E. Selection

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. Two types of selection are used in this research, which are:

E.1 Rank-weighted roulette wheel

This approach uses a uniform random number generator to select chromosomes. The row numbers of the parents are found using:

ma = ceil ($\mathbf{N_{keep}}$ * rand(1, $\mathbf{N_{keep}}$/2))
pa =ceil($\mathbf{N_{keep}}$ * rand(1, $\mathbf{N_{keep}}$/2)), Where *ceil* rounds the value to the next highest integer.

This approach is problem independent and finds the probability from the rank of the chromosome. Rank weighting is slightly more difficult to program than the other selection types. Small populations have a high probability of selecting the same chromosome. The probabilities only have to be calculated once. We tend to use rank weighting because the probabilities don't change each generation.

E.2 Tournament selection

Another approach that closely mimics mating competition in nature is to randomly pick a small subset of chromosomes (two or three) from the mating pool, and the chromosome with the lowest cost in this subset becomes a parent. The tournament repeats for every parent needed. Thresholding and tournament selection make a nice pair, because the population never needs to be sorted. Tournament selection works best for large population sizes because sorting becomes time-consuming for large populations. 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 *roulette-wheel* and *tournament* parent selection procedures [4].

F. Crossover

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 real-coded 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 these point crossover methods is that no new information are 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 methods [4, 5] remedy this problem by finding ways to combine variable values from the two parents into new variable values in the offspring. A single offspring variable value ($\mathbf{b_{new}}$) comes from a combination of the two corresponding parent variable values:

$$b_{new}=\beta b_{mn}+(1-\beta)b_{dn} \quad (7)$$

Where, $\boldsymbol{\beta}$ = random number on the interval [**0, 1**], $\mathbf{b_{mn}}$ = n^{th} variable in the mother chromosome, $\mathbf{b_{dn}}$ = n^{th} variable in the father chromosome.

The same variable of the second offspring is merely the complement of the first (*i.e.* replacing $\boldsymbol{\beta}$ by $\mathbf{1-\beta}$). If $\boldsymbol{\beta = 1}$, then ($\mathbf{b_{mn}}$) propagates in it's entirely and ($\mathbf{b_{dn}}$) dies. In contrast, if $\boldsymbol{\beta = 0}$, then ($\mathbf{b_{dn}}$) propagates in it's entirely and ($\mathbf{b_{mn}}$) dies. When $\boldsymbol{\beta = 0.5}$, the result is an average of the variables of the two parents. This method demonstrated to work well on several interesting problems. Choosing which variables to blend is the next issue. Sometimes, this linear combination process is done for all variables to the right or to the left of some crossover point. Any number of points can be chosen to blend, up to ($\mathbf{N_{var}}$) values where all variables are linear combinations of those of the two parents. 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
$\boldsymbol{\beta > 1}$.

G. Mutation

Random mutations alter a certain percentage of the genes in the list of chromosomes. 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. Mutation points are randomly selected from the ($\mathbf{N_{ind}}\times\mathbf{N_{var}}$), total number of genes in the population matrix. Increasing the number of mutations increases the algorithm's freedom to search outside the current region of variable space. It also tends to distract the algorithm from converging on a popular solution. With the process of the crossover and mutation taking place, there is a high chance that the optimum solution could be lost as there is no guarantee that these operators will preserve the fittest string. To counteract this, *elitist* models are often used. In an elitist model, the best individual in the population is saved before any of these operations take place. After the new population is formed and evaluated, it is examined to see if this best structure has been preserved. If not, the saved copy is reinserted back into the population. The genetic algorithm then continues on as normal [6].

IV. MULTIPLE OBJECTIVE OPTIMIZATION (MOO)

In many applications, the cost function has multiple, often times, conflicting objectives. The most important approach to MOO is: sum of weighted cost functions. The most straightforward approach to multi-objective optimization is to weight each function and add them together.

$$\text{cost}=\sum_{i=1}^{N} w_i f_i \quad (8)$$

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

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

Implementing this multiple objective optimization approach in a genetic algorithm only requires modifying the cost function to fit the form of equation (8) and does not require any modification to the genetic algorithm. Thus,

$$\text{cost}=wf_1+(1-w)f_2 \quad (9)$$

In load flow problem, (f_1) is the mismatch active power (eqn.3) and (f_2) is the mismatch reactive power (eqn.4) for each bus except the slack busbar. This approach is adopted in this work for its simplicity, ease of programming and gives us the required accuracy. Here, (**w**) is chosen to be (**0.5**), since the two objective functions (f_1) and (f_2) have the same degree of importance [7].

V. SPARSITY TECHNIQUES

Sparse matrices are a special class of matrices that contain a significant number of zero-valued elements. This property allows to:

•Store only the nonzero elements of the matrix, together with their indices, to reduce the storage requirements.
•Reduce the computation time for any arithmetic operation by eliminating operations on zero elements [8].

A. Sparse matrix storage

For full matrices, any software package stores internally every matrix element. Zero-valued elements require the same amount of storage space as any other matrix element. For sparse matrices, however, the sparsity technique stores only the nonzero elements and their indices. For large matrices with a high percentage of zero-valued elements, this scheme significantly reduces the amount of memory required for data storage. The implementation of sparsity technique, for example in MATLAB uses three arrays internally to store sparse matrices with real elements. Consider an (***m*-by-*n)*** sparse matrix with (**NNZ**) nonzero entries (NNZ is number of nonzero elements):

•The first array contains all the nonzero elements of the array in floating-point format. The length of this array is equal to (**NNZ**).
•The second array contains the corresponding integer row indices for the nonzero elements. This array also has length equal to (**NNZ**).
•The third array contains integer pointers to the start of each column. This array has length equal to (**n**).

This matrix requires storage for (**NNZ**) floating-point numbers and (**NNZ+n**) integers. At 8 bytes per floating-point number and 4 bytes per integer, the total number of bytes required to store a sparse matrix is:

Grand total of bytes=8*NNZ+4*(NNZ+n) (10)

Sparse matrices with complex elements are also possible. In this case, it uses a fourth array with (**NNZ**) elements to store the imaginary parts of the nonzero elements. An element is considered nonzero if either its real or imaginary part is nonzero.

B. Creating Sparse Matrices

Every software package never creates sparse matrices automatically. Instead, we must determine if a matrix contains a large enough percentage of zeros to benefit from sparse techniques. The ***density*** of a matrix is the number of nonzero elements divided by the total number of matrix elements. Matrices with very low density are often good candidates for use of the sparse format. In contrast, the ***matrix sparsity*** is the number of zero elements divided by the total number of matrix elements. Matrices with very high matrix sparsity are often good candidate for use of the sparse format.

C. Viewing sparse matrix

We can provide a number of functions that let us get quantitative or graphical information about sparse matrices. The MATLAB's commands provide high-level information about matrix storage, including size and storage class. For example, the following list shows information about sparse and full versions of the same matrix:

Illustration example:

Name	Size	Bytes	Class
M_full	(1100x1100)	9680000	double array
M_sparse	(1100x1100)	4404	sparse array

Grand total is (1210000) elements using (9684404 bytes). Notice that the number of bytes used is much less in the sparse case, because zero-valued elements are not stored. In this case, the density of the sparse matrix is (4404/9680000), or approximately 0.00045 (0.045%).

VI.. THE PROPOSED METHOD: MULTI-OBJECTIVE REAL-CODED GENETIC ALGORITHM WITH SPARSITY TECHNIQUE (IMPLEMENTATION AND RESULTS)

Three test systems were used to demonstrate the performance of the proposed method, namely:

1. 14-bus IEEE International test system which consists of: 1 slack bus, 4 generator busses (**PV**) and 9 load buses (**PQ**), with 20 branches [3].
2. 30-bus IEEE test system [3], which consists of:1 slack bus, 5 generator buses (**PV**) and 24 load buses (**PQ**), with 41 branches.
3. Iraqi National Grid (ING), which consists of **362 busbars**.1 slack bus, 29 generator bus (**PV**) and 332 load bus (**PQ**), with 599 branches.

The load flow solution using real-coded genetic algorithm programs with and without sparsity technique have been developed by the use of MATLAB version7, and tested with a Pentium 4, 3GHz (Cache 2M) PC with 2GB RAM. Table I illustrates the power flow solution for 14-bus IEEE test system using RCGA with sparsity technique with two objective functions which are the

mismatch active and reactive powers at each bus according to its constraints except the slack bus. The sum of weighted cost functions is used. 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 is shown in the table. The total computation time was 4.15 sec while the total computational time without using sparsity technique was 7.156 sec with the same accuracy (cost function ≤ 0.001 p.u., corresponding to 0.1 MW/MVAr). Table II illustrates the reduction in computational time and storage requirements for different power systems by using the proposed method. The reduction in computational time and storage requirements increase as matrix density decreases or in other words matrix sparsity increases.

VII. CONCLUSIONS

The proposed method (RCGA with sparsity technique) presented in this paper is much faster and has less storage requirements than the simple genetic algorithm. Thus it can be concluded that the proposed method can be implemented on-line for small and medium-scale power systems 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.

Table I. Power flow solution of 14-bus IEEE test system by the RCGA with sparsity technique method with a standard accuracy of (cost function ≤ 0.001 p.u.)

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	0.000329	PV	1.045	-3.2117	15
3	0.000131	PV	1.00	-4.35826	6
4	0.000484	0.000689	1.041	-6.14362	19
5	0.000890	0.000113	1.045	-12.4235	40
6	0.000798	PV	1.07	6.30252	75
7	0.000365	0.000324	1.071	-4.6541	87
8	0.000222	PV	1.09	-1.71203	101
9	0.000185	0.000605	1.06	1.44081	72
10	0.000273	0.000321	1.063	-9.00316	18
11	0.000950	0.000221	1.063	-5.48283	90
12	0.000411	0.000536	1.061	7.67754	43
13	0.000770	0.0005215	1.059	-11.0288	29
14	0.000209	0.000765	1.056	-3.34466	47
***Total**	**Computational**	**Time:**			**4.15 sec.**

*Total Computational Time using RCGA without Sparsity Technique = 7.156 sec.

Type of power system	Matrix density of [Y]	%Reduction in Computation Time Enhanced RCGA	%Reduction in Storage Requirement Enhanced RCGA
14-bus IEEE	17.34%	97.48%	60%

30-bus IEEE	7.88%	94.78%	75%
362-bus(ING)	0.628%	99.99%	95%

Table II. Comparison of reduction in computation time and storage requirement for different power systems using The RCGA with sparsity technique method

VIII. REFERENCES

[1] N. R. Dhar, "Computer Aided Power System Operation and Analysis", Department of Electrical Engineering, Jadavpur University, Calcutta, 1982.

[2] L. Ippolito, A. Cortiglia, and M. Petrocelli, "Optimal Allocation of Facts Devices by Using Multi-Objective Optimal Power Flow and Genetic Algorithms", *International journal of emerging electric power systems*, Vol. 7, No. 2, 2006.

[3] H. A. Kubba, "A improved and more reliable decoupled load flow method", Engineering, Scientific Journal of Engineering college/Baghdad University, No.3, Vol. 7, September, 2001 pp. 25-37.

[4] M. Mitchell, "An Introduction to Genetic Algorithms", MIT Press, Cambridge, Massachusetts-London, England, 5th printing, 1999.

[5] H. T. Yang, and L.C. Huang, "A Parallel Genetic Algorithm Approach to Solving the Unit Commitment Problem", *IEEE Transactions on power systems*, Vol. 12, No. 2, May 1997.

[6] S.B.M. Ibrahim,"The PID Controller Design Using Genetic Algorithm", A dissertation submitted to University of Southern Queensland, Faculty of engineering and surveying,Electrical and Electronics Engineering, October, 2005

[7] A. Talib, "An Optimization Approach of Robot Motion Planning Using Genetic Algorithm", M.Sc Thesis, Mechatronics Department, AL-Khwarizmi Engineering, University of Baghdad, 2007.

[8] A. Brameller, R. N. Allan and Y. M.Haman, "Sparisty ", Pitman Publishing, 1976.