An Introduction to Genetic Algorithms: A survey A practical Issues

Ahmed A. EL- Sawy[1, 2], Mohamed A. Hussein[1], EL-Sayed M. Zaki[1], A. A. Mousa[1, 3]

[1] Department of Basic Engineering Sciences, Faculty of Engineering, Menoufia University, Shibin El-Kom, Egypt.
[2] Department of Mathematics, Faculty of Science, Qassim University, Saudi Arabia.
[3] Department of Mathematics, Faculty of sciences, Taif University, Saudi Arabia.

Abstract— The Genetic Algorithm (GA) is a relatively simple heuristic algorithm that can be implemented in a straightforward manner. It can be applied to a wide variety of problems including unconstrained and constrained optimization problems, nonlinear programming, stochastic programming, and combinatorial optimization problems. It is widely used in several fields such as management decision making, data processing ...Information and Financial Engineering. Because of their population approach, they have also been extended to solve other search and optimization problems efficiently, including multimodal, multiobjective. In this paper, a brief description of a simple GA, GAs vs. traditional methods and GAs to handle constrained optimization problems are described. Also, GAs for multiobjective optimization MOP is proposed. Thereafter, GAs applications are presented. The intended audience of this paper is those who wish to know the main concepts of GAs and how to apply it to different optimization problems. Also, to familiarize readers to the algorithm proceeding.

Index Terms— Genetic algorithms; constrained Optimization; Multiobjective optimization.

1 INTRODUCTION

Genetic Algorithms (GAs) are apart of the evolutionary algorithms, which is a rapidly growing areas of artificial intelligence [1]. GAs are inspired by Darwin's theory of biological evolution. By mimicking this process, genetic algorithm are able to "evolve" solutions to real world problems. Holland [2] was the first person to put computational evolution on a firm theoretical footing. GAs are optimization algorithms based on the concepts of biological evolution and genetics. In this algorithm, the design variables are represented as genes on a chromosome. GAs feature a group of candidate individuals (which is called population) on the response surface. Through environmental selection and the genetic operators, mutation and recombination, chromosomes with better fitness are found. Natural selection guarantees that best chromosomes with better fitness will survive in the future populations. Using the recombination operator the GA combines genes from two parent chromosomes to form children (new chromosomes) that have a high probability of having better fitness than their parents. On the other hand, mutation allows new areas of the response surface to be explored. GAs offer a generation improvement in the fitness of the chromosomes and after many generations will create chromosomes containing the optimized variable settings [3],[4],[5],[6],[7].

Genetic algorithm was invented by "John Holland" in the 1960s and it was later developed by Holland and his students and colleagues at the University of Michigan in 1960s and 1970s. Holland's 1975 book" Adaptation in Natural and Artificial Systems"[2],[7] presented the genetic algorithms as an abstraction of biological evolution and gave a theoretical framework for adaptation under the genetic algorithms. Holland's original goal was not to design an algorithm to solve specific problems, but rather to formally study the phenomenon of adaptation as it occurs in nature and to develop ways in which the mechanisms of natural adaptation might be imported to computer systems [7],[8].

2. THE BASIC GENETIC ALGORITHMS

The basic or simple GA comprises four important steps [5]:

Step 1: The initial candidate population of chromosomes is created either randomly or by perturbing an input chromosome. Indeed, how the initialization step is done is not critical as long as the initial population spans a wide range of design variable settings. Thus, if you have explicit knowledge about the system being optimized that information can be included in the initial population [7]. In the binary representation, every chromosome is a string of bits, 0 or 1. The length of the string depends on the required precision (number of decimal places). Suppose that each variable xi can take values from the domain $D_i = [a_i, b_i] \subseteq R$: suppose Q decimal places for the variables values is desirable. It is clear to achieve such precision each domain D_i should be cut into $(b_i - a_i).10^Q$ equal size ranges. Let mi be the smallest integer such that $(b_i - a_i).10^Q \le 2^{m_i} - 1$.Then a representation (Fig.1) having each variable xi coded as a binary string of length mi additionally, the following formula interprets each such string:

$$x_i = a_i + decimal(1001.....001_2).\frac{b_i - a_i}{2^{m_i} - 1}. \qquad (1)$$

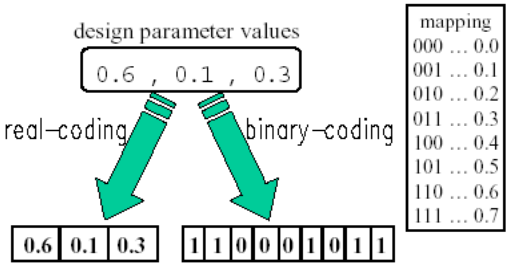


Fig. 1: An example of binary and floating-point representations

Step 2: Evaluation, the fitness is computed this step,. The goal of the fitness function is to numerically encode the performance of the chromosome. For real-world applications of optimization methods such as evolutionary algorithms the choice of the fitness function is the most critical step [7].

Step 3: In this step, the chromosomes with the largest fitness scores are placed one or more times into a mating pool subset in a semi-random fashion. Chromosomes with low fitness are removed from the population. There are several methods for performing selection. One of the most common methods roulette wheel selection where every chromosome has its place big accordingly to its fitness function, this can be simulated by following algorithm (Fig. 2).

1.[SUM] *Calculate sum of all chromosomes fitness* (f_i) *in population*, $F = \sum_{i=1}^{pop-size} f_i$.

2. [COMPUTE] *Compute probability of selection for each individual* i *as,* $p_i = \frac{f_i}{F}$.

3. [Loop] *Go through the population from* i**=1** *to pop-size (N).*

- **[COMPUTE]** *Compute cumulative probability for each* i *as,* $Q_i = \sum_{j=1}^{i} p_j$
- **[SELECT]** *Generate random number from* $r \in [0,1]$,
- **[DECISION]** *When* $Q_{i-1} \le r_i \le Q_i$ *select chromosome* i .

Fig .2: Roulette Wheel Algorithms

Step 4: Exploration, consists of the crossover and mutation operators. Two chromosomes (i.e., parents) from the mating pool subset are randomly selected to be mated. The probability that these parents are recombined (mated) is a user-controlled option and is usually set to a high value. If the parents are allowed to mate, a crossover operator is employed to exchange genes between the two parents to produce two offspring. If they are not allowed to mate, the parents are copied into the next generation unchanged. The two most common recombination operators are the one-point and two-point crossover methods. In the one-point method, a crossover point is selected along the chromosome and the genes up to that point are swapped between the two parents. However, in the two-point method, two crossover points are selected and the genes between the two points are swapped. The children then replace the parents in the next generation. A third recombination operator, which has become quite popular, recently, is the uniform crossover method. In this crossover method, recombination is applied to the individual genes in the chromosome. If crossover is performed, the genes between the parents are swapped and if no crossover is performed the genes are left intact. This crossover method has a higher probability of producing children which are much different than their parents so the probability of recombination is usually set to a low value. The probability that a mutation will occur is another user-controlled option and is usually set to a low value so that good chromosomes are not destroyed. A mutation simply changes the value for a particular gene. After the exploration step, the population is full of newly created chromosomes (children) and steps two through four are repeated. This process continues for a fixed number of generations [7]. Figure 3 shows a flowchart of the working of a GA.

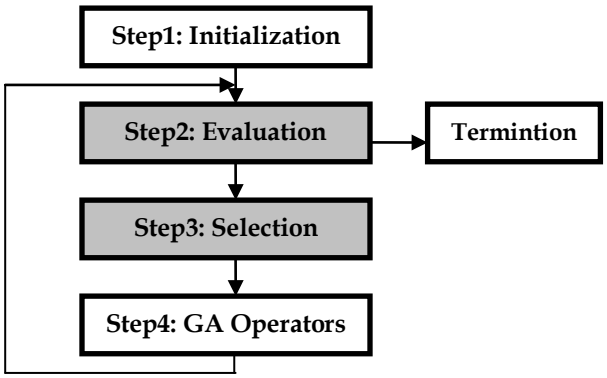


Fig. 3: Main Flowchart Of SGA

3. Genetic Algorithms Vs Traditional Methods

Traditional methods [9],[10],[11],[12]

Calculus-based methods: The main disadvantages of calculus-based search are, firstly, a tendency for the search to get trapped on local maxima even a though a better solution may exist , all moves from local maxima seem to decrease the fitness of the solution, secondly the application of such searches depends on the existence of derivatives.

Dynamic programming: This optimization technique builds towards a solution by first solving a small part of the whole problem, and then gradually incrementing the size in a series of stages until the whole problem is solved, This is a method for solving multistage control problems, but can only use where the number of state and stages are small, and there is no interaction between stages.

Random search: This is a brute force approach to difficult functions, also called an enumerated search. Points in the search space are selected randomly this is a very unintelligent strategy [4],[5].

Gradient methods: Such methods are generally referred to as hill-climbing, and perform well on functions with only one peak. However, on functions with many peaks, the first peak found will be climbed, whether is it the highest peak or not. Finally four difference that separate GAs from conventional optimization techniques are concluded [4],[5],[13].

1- Direct manipulation of a coding. GAs manipulates decision or control variable representations at a string level to exploit similarities among high- performance strings. Other methods usually deal with functions and their control variables directly [14].

2- Search form a population, not a single point. In this way GAs find safety in numbers. By maintaining a population of well adapted sample points, the probability of reaching a false peak is reduced. The search starts from a population of

many points, rather than starting from just one point. This parallelism means that the search will not become trapped on a local maxima especially if a measure of *diversity* maintenance is incorporated into the algorithm, for then one candidate may become trapped on a local maxima, but the need of maintain diversity in the search population means that other candidates will therefore avoid that particular area of the search space.

3- Search via sampling, a blind search. GAs achieves much of their breadth by ignoring information except that concerning payoff. Other methods rely heavily on such information and in problems where the necessary information is not available or difficult to obtain, these other techniques break down [14].

4- Search using stochastic, not deterministic rules. The transition rules used by genetic algorithms are probabilistic, not deterministic.

Finally, the advantages and disadvantages of using GAs are concluded as follows [7]:

- They require no gradient information about the response surface
- Discontinuities present on the response surface have little effect on overall optimization performance
- They are resistant to trapped in local optima
- They perform very well for large scale optimization problems
- Can be employed for a wide variety of optimization problems

Disadvantages to using genetic algorithms

- Have trouble finding the exact global optima.
- Require large number of response (fitness) function evaluations
- GAs configuration is not straightforward

4- WHY DO GAS WORK?

The heuristic search of a GAs is based upon Holland's schema theorem. The mathematics of this theorem were developed using the binary representation [5],[7]. A brief non-mathematical introduction of the schema will be given assuming a binary coding. A schema S is defined as a template for describing a subset of chromosomes with similar partitions. The template consists of multiple 0's, 1's, and don't care symbols (*), this character is simply a notational device used to signify that either a 1 or 0 will match that pattern. Given an example, consider a schema such as, ***0000**. This schema matches two chromosomes, **10000** and **00000**. The template is a powerful way of describing similarities among patterns in the chromosomes. According to Holland, the order of a schema **o(s)** is equal to the number of fixed positions (i.e., non-meta-characters) and the defining length of a schema **δ(s)** is the distance between the first and the last fixed string positions. Thus, the schema **#00#0** is an order 3 schema (**o(s) = 3**) and has defining length of **(δ=5-2=3)**. Holland derived an expression that predicts the number of copies a particular schema, **s**, w=3ould have in the next generation after undergoing exploitation, recombination and mutation. This expression is shown below

$$\zeta(s,t+1) \geq \zeta(s,t).eval(s,t) / \bar{F}(t)[1 - p_c.\frac{\delta(s)}{m-1} - o(s).p_m] \qquad (2)$$

where **s** is a particular schema, t is the generation, $\zeta(s,t+1)$ is the number of times a particular schema is expected in the next generation, $\zeta(s,t)$ is the number of times the schema is in the current generation, $eval(s,t)$ is the average fitness of all chromosomes that contain schema **s**, $\bar{F}(t)$ is the average fitness for all chromosomes, P_c is the probability of crossover occurring, and P_m is the mutation probability. The primary conclusion that can be drawn from inspection of this equation is that as the ratio of $eval(s,t)$ to $\bar{F}(t)$ becomes larger, the number of the times **s** is expected in the next generation increases. Thus, particularly good schemata will propagate in future generations. Two more points need to be made concerning Holland's schema theorem. Although both mutation and recombination destroy existing schemata, they are necessary for building better ones. The degree to which they are destroyed is dependent upon the order (***o(s)***) and the length (**δ**(s)) of the schemata. Thus, schemata that are short, low-order, and have above average fitness are preferred and are termed "building blocks". This definition leads to the building block principle of GAs which states that there is a high probability that short, low-order, average fitness schemata will combine through recombination to form higher order, above average fitness schemata.

5. AN EXAMPLE OF APPLYING GENETIC ALGORITHMS

Here we present a simple example of applying genetic algorithm, which taken from [6].

Min

$$Z = [30 + (2x - 3y)^2(18 - 32x + 12x^2 + 48y - 36xy + 27y^2)] *$$
$$[1 + (x + y + 1)^2(19 - 14xy + 3x^2 - 14x + 6xy + 3y^2)]$$

s.t.

$$-2.0 \leq x \leq 2.0, \quad -2.0 \leq y \leq 2.0$$

5.1 Representation

First, decision variables must be encode into binary strings. Here 16 bits (Suppose that 4 decimal places for the variables values is desirable) are used to represent a variable. The mapping from a binary string to a real number for a variable x or y is computed from equation 1 as follows:

$$x = -2.0 + x' \cdot \frac{4.0}{2^{16} - 1}, \qquad y = -2.0 + y' \cdot \frac{4.0}{2^{16} - 1}$$

Here x' and y' represent the decimal value of the substring for decision variable x and y respectively.

5.2 Initial Population

In each generation the population size is set as 20. Initial population is randomly generated as follows:

v_1 = [11100110110011000110101110000010] = [1.606256, −0.320165] F=2907.700817; EVAL=700242.428
v_2 = [11000000001100011011110001010101] = [1.003037, 0.942733] F=1470.604014; EVAL=701679.525
v_3 = [11000001010000101011010100011000] = [1.019699, 0.829633] F=998.466596; EVAL=702151.663
v_4 = [01001000111100011001110110001100] = [−.860273, 0.461707] F=9680.870631; EVAL=693469.258
v_5 = [00110011011111100101110110001010] = [−1.195422, −0.538430] F=1439.880786; EVAL=701710.248
v_6 = [01000110111110000000011100011000] = [−.891096, −1.881346] F=11273.574224; EVAL=691876.555
v_7 = [01101000000111101000011001111010] = [−.373144, .101228] F=951.091206; EVAL=702199.038
v_8 = [00010110000101111010111111110000] = [−1.654841, .749065] F=8068.650332; EVAL=695081.479
v_9 = [10110111110111111010110000111111] = [.873030, 0.691386] F=982.663173; EVAL=702167.466
v_{10} = [00000000010111111100110101111111] = [−1.1994202, 1.210925] F=45585.613158; EVAL=657564.516
v_{11} = [10011001101010100000111011010100] = [.41038, −1.768307] F=8488.275825; EVAL=694661.853
v_{12} = [00111000000010011111101010110010] = [−1.124437, 1.917174] F=703150.129106; EVAL=0.000
v_{13} = [01110111110010101111101101101011] = [−0.128267, 1.928466] F=234882.112971; EVAL=468268.016
v_{14} = [00010110110111111001000101111110] = [−1.642634, −1.453239] F=14013.064752; EVAL=689137.064
v_{15} = [10001101001001001110011110001110] = [0.205356, 1.613443] F=84257.482260; EVAL=618892.647
v_{16} = [10010100011101110010000111001000] = [0.319799, −1.472160] F=730.102530; EVAL=702420.027
v_{17} = [11110010010010111010011101000010] = [1.785885, 0.618090] F=890.983919; EVAL=702259.145
v_{18} = [11011001111111110101110001000111] = [1.406241, −.558145] F=5332.051371; EVAL=697818.078
v_{19} = [00000110101010100001101111110001] = [−1.895872, −1.563409] F=21833.496910; EVAL=681316.632
v_{20} = [11011010100100101000011010000110] = [1.457633, −.948836] F=26032.543455; EVAL=677117.586

5.3 Evaluation

The first step after creating a generation is to calculate the fitness value (**F**) of each member in the population by calculating the objective function for each individual. The process of evaluating the fitness of a chromosome consists of the following three steps:

1. Convert the chromosome's genotype to its phenotype. This means converting the binary string into corresponding real values as in equation 1.

2. Evaluate the objective function, which we refer as the fitness value (**F**)

3. Convert the value of objective function into fitness. Here, in order to make fitness values positive,(the positive values needed for Roulette Wheel Algorithm) the fitness of each chromosome (**EVAL**) equals the maximization of the objective function minus the objective function evaluated for each chromosome in the population. The objective function values **F** and the fitness values **EVAL** of above chromosomes (the first population) are depicted before. It is clear that in the first generation chromosome V16 is the **best** one (EVAL=702420.027) and that chromosome V12 is the **poorest** one (EVAL=0.000)

5.4 Create a new population

After evaluation, a new population should be created from the current generation. Here the three operators (Elitism, Crossover, and Mutation) are used.

5.4.1 Elitism

The two chromosomes (strings) with best fitness are allowed to survive and produce offspring in the next generation. For example, in first population, chromosome V16 and V17 are allowed to live in the second generation.

5.4.2 Selection and Crossover

The cumulative probability is used to decide which chromosomes will be selected to crossover. The cumulative probability is calculated as in figure 2.

$P_1 = 0.054$ $Q_1 = 0.054$ $P_2 = 0.054$ $Q_2 = 0.109$ $P_3 = 0.055$ $Q_3 = 0.163$
$P_4 = 0.054$ $Q_4 = 0.217$ $P_5 = 0.054$ $Q_5 = 0.272$ $P_6 = 0.054$ $Q_6 = 0.325$
$P_7 = 0.055$ $Q_7 = 0.380$ $P_8 = 0.054$ $Q_8 = 0.434$ $P_9 = 0.055$ $Q_9 = 0.488$
$P_{10} = 0.014$ $Q_{10} = 0.539$ $P_{11} = 0.054$ $Q_{11} = 0.593$ $P_{12} = 000$ $Q_{12} = 0.593$
$P_{13} = 0.036$ $Q_{13} = 0.630$ $P_{14} = 0.054$ $Q_{14} = 0.683$ $P_{15} = .048$ $Q_{15} = 0.731$
$P_{16} = 0.055$ $Q_{16} = .786$ $P_{17} = 0.055$ $Q_{17} = 0.840$ $P_{18} = 0.054$ $Q_{18} = 0.895$
$P_{19} = 0.053$ $Q_{19} = 0.947$ $P_{20} = 0.053$ $Q_{20} = 1.00$

The crossover used here is one-cut-point method, which randomly selects one cut-point and exchanges the right parts of two parents to generate offspring with $P_c = 1$.

1. Generate a random number *r* from the range [0,1];

2. If $Q_{i-1} \le r_i \le Q_i$ select the *i*th chromosome Vi to be parent one.

3. Repeat step 1 and 2 to reproduce another parent.

4. Generate a random number *r* from the range [0,1]. If *r* is less than the probability of crossover (the probability of crossover as 1.0), the crossover will undergoes, the cut-point is selected behind the gene which place is nearest integers greater than or equal to *r*×(length-1).

5. Repeat step 1 to step 4 altogether **nine** times to finish the whole crossover.

The creation of 18 offspring plus 2 chromosomes obtained using elitism strategy keeps the population the same in each generation in this case 20.

5.4.3 Mutation

Mutation [5] is performed after crossover. Mutation alters one or more genes with a probability equal to the mutation rate. (In the example, the mutation rate is set to $p_m = 0.02$)

1. Generate a sequence of random numbers r_k *(k=1,.....,640)* (Here, the numbers of bits in the whole population is 20×(16+16)=640).

2. If *ri* is 1, change the *i*th bit in the whole population from 1 to 0 or from 0 to 1.

3. The chromosomes reproduced are not subject to mutation, so after mutation, they should be restored. A new population is created as a result of completing one iteration of the genetic algorithm. The procedure can be repeated as many times as desired. In this example, the test run is terminated after 50 generations.

The best value of the objective function in each generation is reported by [6] as follows:

Generation 1: f (0.319799, -1.472160)=730.102530
Generation 2: f (0.406165, -0.558145)=162.226980
……..
Generation 49: f (-0.005158, -0.999924)=3.006779
Generation 50: f (-0.005158, -0.999924)=3.006779

By using GAs in [6] they finally guarantee that GAs converge to the optimal solution after 50 generation

6- CONSTRAINED OPTIMIZATION USING GAS.

The above discussion of optimization problems avoid detailed consideration of constrains. On the other hand, many

real-world optimization problems involve inequality and/or equality constraints are thus posed as constrained optimization problems. In trying to optimize constrained optimization problems using genetic algorithms (GAs) or classical optimization methods, penalty function methods have been the most popular approach. However, since the penalty function approach is generic and applicable to any type of constraint, their performance is not always satisfactory. Thus, several methods for handling unfeasible solutions have emerged recently. General form of the nonlinear programming problem (NLPP) can be defined as follows [15]:

NLPP:

Max $f(\bar{x})$

Subject to:

$$\mathsf{F} = \{x \in R^n \mid g_i(x) \le 0,\ i = 1,2,...,k \text{ and } h_j(x) = 0\ j = k+1,...,m\}$$
$$\mathsf{S} = \{x \in R^n \mid l(x_i) \le x_i \le u(x_i),\quad i = 1,2,.....,n\} \qquad (3)$$

Where $\bar{x} \in \mathsf{F} \subseteq \mathsf{S}$. The set $\mathsf{S} \subseteq R^n$ defines the search space and the set $\mathsf{F} \subseteq \mathsf{S}$ defines a feasible part of the search space. Usually, the search space S is defined as *n*-dimensional rectangle in R^n (domains of variables defined as lower and upper bounds): $left(i) \le x_i \le right(i),\ 1 \le i \le n$ Whereas the feasible set F is defined by the search space S and an additional set of constraints:

$$g_i(x) \le 0,\ i = 1,2,...,k \text{ and } h_j(x) = 0\ j = k+1,...,m$$

One of the major components of any evolutionary system is the evaluation function. Evaluation functions are used for assign a quality measure for individuals in a population. Whereas evolutionary computation techniques assume the existence of an (efficient) evaluation function for feasible individuals, there is no uniform methodology for handling (i.e., evaluating) unfeasible ones. The simplest approach, incorporated by evaluation strategies and the version of evolutionary programming (for numerical optimization problems), is to reject unfeasible solutions. But several other methods for handling unfeasible individuals have emerged recently.

6.1 Methods Based on Penalty Functions [15],[16],[17]

The penalty function method is widely used in the mathematical Programming literature. It essentially adds to the objective function some terms which punish a solution that is not feasible.

the above NLPP (3) can be transformed into an unconstrained optimization problem. The objective function of the unconstrained optimization problem, which will be used as the fitness function in the associated genetic algorithm designed to solve the initial constrained problem, has the following format:

$$eval(x) = \begin{cases} f(x), & if\ x \in \mathsf{F} \\ f(x) + penalty(x), & otherwise \end{cases} \qquad (4)$$

where *penalty(x)* is zero, if no violation occurs, and is positive, otherwise. Usually, the penalty function is based on the distance of the solution form the feasible region F, or on the effort to "repair" the solution, i.e., to force it into F. The former case is the most popular one; in many methods a set of functions $f_j (1 \le j \le m)$ is used to construct the penalty, where the function $f_j(x)$ measures the violation of the *j-th* constraint in the following way:

$$f_j(x) = \begin{cases} max\ \{0, g_j(x)\} & if\ 1 \le j \le k \\ \left|h_j(x)\right| & if\ k+1 \le j \le m \end{cases} \qquad (5)$$

How the penalty function is designed and applied to unfeasible solutions may differ in important details across problems.

6.1.1 Static Penalty Function

The static penalty function assumes that for every constraint we establish a family of intervals which determine an appropriate penalty coefficient R_{ij}. It works as follows: (1) for each constraint, create several (*l*) levels of violation (these levels measure the degree of violation, e.g., slightly or heavily); (2) for each level of violation and for each constraint, create a penalty coefficient $R_{ij} (i = 1,2,...,l,\ j = 1,2,...,m)$; higher degree of violation requires heavier punishment (i.e., larger R_{ij}). The evaluation function has the following structure:

$$eval(x) = f(x) + \sum_{j=1}^{m} R_{ij} f_j^2(x) \qquad (6)$$

Where the $f_j(x)$ are as defined above and *m* is the number of constraints in the problem the central issue in this method is the determination of the relative magnitudes of the coefficients $\{R_{ij} \mid 1 \le i \le l, 1 \le j \le m\}$.

The weakness of the method is in the number of parameters. For *m* constraints the method requires *m(2l+1)* parameters in total: *m* parameters to establish number of intervals for each constraint, *l* parameters for each constraint, defining the boundaries of the intervals (levels of violation), and *l* parameters for each constraint representing the penalty coefficient R_{ij}.

6.1.2 Dynamic Penalty Function

Dynamic penalty function method differs from the previous one in that it punishes "harder" as the number of generations increases. The implementation of this method is through the following evaluation function such that individuals are evaluated (at the iteration *t*) by the following formula:

$$eval(x) = f(x) + (C*t)^{\alpha} \sum_{j=1}^{m} f_j^{\beta}(x) \qquad (7)$$

Where C, α and β is a constant. A reasonable choice for these parameters is $C = 0.5, \alpha = \beta = 2$

i.e.,

$$eval(x) = f(x) + (0.5t)^2 \sum_{j=1}^{m} f_j^2(x) \qquad (8)$$

The method requires much smaller number of parameters than the first method. Also, instead of defining several levels of violation, the pressure on unfeasible solutions is increased due to the $(C*t)^{\alpha}$ component of the penalty term: towards the end of the process (for high values of the generation number t), this component assumes large values.

6.1.3 Rejection of Unfeasible Individuals

This "death penalty functions" method is a popular option in many evolutionary techniques like evolutionary strategies or evolutionary programming, this method rejects all unfeasible solutions in the population. Thus, under this method, if in some current population unfeasible solutions result after the GA operators are applied; these are simply eliminated and replaced by randomly drawn new solutions.

6.2 Behavioral Memory Method [16],[18]

The idea of this technique is to satisfy sequentially (one to one) the constraints imposed on the problem. Once a certain percentage of the population (defined by the flip threshold) satisfies the first constraint, an attempt to satisfy the second constraint (while still satisfying the first) will be made. Notice that in its last step of the algorithm, death penalty was used, because unfeasible individual are completely eliminated from the population.

6.3 Repair Methods [16],[18]

Repair algorithms enjoy a particular popularity in the evolutionary computation community. GENOCOP III [16] is based on the idea of repairing unfeasible solutions, and its algorithm needs at least one feasible point to enter the evolution process. In this algorithm any unfeasible point must be repaired to become feasible one. The weakness in this algorithm is locating such a reference point especially when the problem have small F/S for the purpose of initialization. So the major difference between constraint and unconstraint optimization is the evaluation function , that is, how to handle unfeasible solution thus the flowchart of SGA are still as it for constrained optimization but step2 will be modified to handle both feasible and unfeasible solutions.

7- MULTIOBJECTIVE OPTIMIZATION

In a multiobjective optimization problem MOP, there are more than one objective functions, which are to be optimized simultaneously. Traditionally, the practice is to convert multiple objectives into one objective function (usually a weighted average of the objective is used) and then to treat the problem as a single objective optimization problem. Unfortunately this techniques is subjective to the user, with the optima solution being dependent on the chosen weight vector. In fact, the solutions of the multiobjective optimization problem can be thought as a collection of optimal solutions obtained by solving different single objective functions formed using different weight vectors. these solutions are known as Pareto optimal solutions[19]. Therefore, the optimization goal for an MOP may be reformulated in a more general fashion based on three objectives:

- The distance of the resulting nondominated front to the Pareto-optimal front should be *minimized*.
- A good (*in most cases uniform*) distribution of the solutions found is desirable.
- The spread of the obtained nondominated front should be maximized, i.e., for each objective a wide range of values should be covered by the nondominated solutions.

The subject of here is the question of how these subgoals can be attained in evolutionary multiobjective search. After the basic terminology, fundamental ideas of MOEAs are introduced in the following section, where in particular the differences between evolutionary single-objective and multiobjective optimization are worked out. Then, a brief summary of three salient evolutionary approaches to multiobjective optimization is presented.

7.1 Basic Definitions and Concept

Here we introduce some of the basic terminology used in the field of evolutionary Algorithms for Multiobjective Optimization.

Local Pareto-optimal Set: If for every member x in a set $\overline{p}$, there exist no solution y satisfying $\|y-x\|_\infty \le \varepsilon$, where ε is a small positive number (in principle, y is obtained by perturbing x in a small neighborhood), which dominates any member in the set $\overline{p}$, then the solutions belonging to the set $\overline{p}$ constitute a local Pareto-optimal set.

Global Pareto-optimal Set: If there exists no solution in the search space which dominates any member in the set $\overline{p}$, then the solutions belonging to the set $\overline{p}$, constitute a global Pareto-optimal set.

A solution $x^{(1)}$ is said to dominate the other solution $x^{(2)}$, if both the following conditions are true [20] :

1. The solution $x^{(1)}$ is no worse (say the operator $\prec$ denotes worse and $\succ$ denotes better) than $x^{(2)}$ in all objectives, or $f_j(x^{(1)}) \nprec f_j(x^{(2)})$ for all $j = 1,.....,q$ objectives.
2. The solution $x^{(1)}$ is strictly better than $x^{(2)}$ in at least one objective, or $f_j(x^{(1)}) \succ f_j(x^{(2)})$ for at least one $j \in \{1,.....,q\}$

Genetic drift one of the problems of genetic algorithms for solving multimodal function is that the finite population will eventually converge to only one optimum, due to stochastic errors in the selection process. This phenomena is known as genetic drift.

A niche is a group of individuals which have similar fitness. Normally in multiobjective and multimodal optimization, a technique called **sharing** is used to reduce the fitness of those individuals who are in the same niche, in order to prevent the population to converge to a single solution, so that stable subpopulations can be formed, each one corresponding to a different objective or peak (in a multimodal optimization problem) of the function.

Fitness sharing [4] is the technique used to maintain population *diversity*, which is the most frequently used technique, aims at promoting the formulation and maintenance of stable subpopulations (niches). It is based on the idea that individuals in a particular niche have to share the available resources. *The more individuals are located in the neighborhood of a certain individual, the more its fitness value is degraded*. The neighborhood is defined in terms of a distance measure *d(i, j)* and specified by the so-called niche radius σ_{share}. Mathematically, the shared fitness *F(i)* of an individual i is equal to its old fitness $F'(i)$ divided by its *niche count* m_i :

$$F(i) = \frac{F'(i)}{m_i} = \frac{F'(i)}{\sum_{j=1}^{pop\_size} sh(d(i,j))}$$

An individual's niche count is the sum of sharing function (sh) values between itself and the individuals in the population. A commonly-used sharing function is

$$sh(d(i,j)) = \begin{cases} 1 - \left(\frac{d(i,j)}{\sigma_{share}}\right)^{\alpha} & if \quad d(i,j) < \sigma_{share} \\ 0 & otherwise \end{cases}$$

Where α is a constant and σ_{share} is the niche radius, fixed by the user at some estimate of the minimal separation desired or expected between individuals. Furthermore, depending on how the distance function $d(i, j)$ is defined, one distinguishes two types of sharing:

1. Fitness sharing in solution space $d(i,j) = \|i - j\|$

$$d(i,j) = \sqrt{\sum_{p=1}^{p} \left(\frac{x_p^{(i)} - x_p^{(j)}}{x_p^u - x_p^l}\right)},$$

where p is the number of variables

2. fitness sharing in objective space

$$d(i,j) = \|f(i) - f(j)\|$$

$$d(i,j) = \sqrt{\sum_{k=1}^{q} \left(\frac{f_k^{(i)} - f_k^{(j)}}{f_k^{\max} - f_k^{\min}}\right)},$$

where q is the number of objectives

where $\|\,.\,\|$ denotes an appropriate distance metric. Currently, most MOEAs implement fitness sharing, e.g., (Hajela and Lin 1992[21]; Fonseca and Fleming 1993[22]; Srinivas and Deb 1994[23]).

7.2 Overview of Evolutionary Techniques

Three of the most salient MOEAs have been chosen for the comparative studies. A brief summary of their main features and their differences is given in the following. For a thorough discussion of different evolutionary approaches to multiobjective optimization, the interested reader is referred to [2],[6],[9],[18],[24].

1-Schaffer's Vector Evaluated Genetic Algorithm (VEGA)

Being aware of the potential GAs have in multiobjective optimization, Schaffer1985 [25] proposed an extension of the simple GA (SGA) to accommodate vector_valued fitness measures, which he called the *Vector Evaluated Genetic Algorithm* (VEGA)The selection step was modified so that, at each generation, a number of subpopulations was generated by performing proportional selection according to each objective function in turn. Thus, for a problem with *q* objectives, *q* subpopulations of size (*N/q*) each would be generated, assuming a population size of *N*. These would then be shuffled together to obtain a new population of size *N* as in figure 4, in order for the algorithm to proceed with the application of crossover and mutation in the usual way

(Fitness assignment and selection in VEGA)
INPUT: p_t **(population)**
Output: p' **(mating pool)**
Step 1: Set i = 1 and mating pool $p' = \phi$
Step 2: For j = 1, . . . , N/q do select individual i from p_t **according to a given scheme and copy it to the mating pool:** $p' = p' + \{i\}$ **.**
Step 3: Set i = i + 1.
Step 4: If i ≤ k then go to Step 2 else stop.

Fig. 4: Fitness assignment and selection in VEGA

This mechanism is graphically depicted in Figure (5a) where the best individuals in each dimension are chosen for reproduction. Afterwards, the mating pool is shuffled and crossover and mutation are performed as usual. Schaffer implemented this method in combination with fitness proportionate selection.

2-Srinivas and Deb's non-dominated sorting genetic algorithm (NSGA)

Using the concept of sharing functions, Srinivas and Deb[26] have implemented Goldberg's idea most directly. The idea behind NSGA is that a ranking selection method is used to emphasize current non-dominated points and sharing function method is used to maintain diversity in the population. The NSGA procedure will be described in somewhat more details.

NSGA varies from a simple genetic algorithm only in the way the selection operator in used. The crossover and mutation operators remain as usual. Before the selection is performed, two procedures are performed serially. First, the population is ranked on the basis of an individual's non-domination level and then sharing function method is used to assign fitness to each individual. We describe both these mechanisms in the following subsections

.

Classifying a population according to non-domination

Consider a set of N population members, each having q (>1) objective function values. The following procedure in figure 5 can be used to find the non-dominated set of solutions:

All these non-dominated solutions are assumed to constitute the first non-dominated front in the population. In order to find the solutions belonging to the second level of non-domination, we temporarily disregard the solutions of the first level of non-domination and follow the above procedure. The resulting non-dominated solutions are the solutions of the second level of non-domination. This procedure is continued till all solutions are classified into a level of non-domination. It is important to realize that the number of different non-domination levels could vary between one to N. Figure 7b shows how the procedure can be used to identify five different levels of non-domination.

Step 0: Begin with i=1.
Step 1: For all j=1,….,N and $j \neq i$,compare solutions $x^{(i)}$ and $x^{(j)}$ for domination using two conditions for all q objectives.
Step 2: If for any j, $x^{(i)}$ is dominated by $x^{(j)}$,mark $x^{(i)}$ as 'dominated'.
Step 3: If all solutions (that is, when i=N is reached) in the set are considered, Go to Step 4, else increment i by one and Go to Step 1.
Step 4: All solutions that are not marked 'dominated' are non-dominated solutions.

Fig. 5: Classifying a population according to non-domination

Fitness assignment

In NSGA, fitness is assigned to each individual according to its non-domination level. *An individual in a higher level gets lower fitness* .This done in order to maintain pressure for choosing solutions from the lower levels of non-domination. Since solutions in lower levels of non-domination are better, a selection mechanism that selects individuals with higher fitness provides a search direction towards the Pareto-optimal region.

First, all solutions in the first non-dominated front n_1 are assigned a fitness equal to the population size *(N)*. This becomes the maximum fitness that any solution can have in any population. Based on the sharing strategy, if a solution has many neighboring solutions in the same front, its dummy fitness is reduced by a factor and a shared fitness is computed. The factor depends on the number and proximity of neighboring solutions. Once all solutions in the first front are assigned their shared fitness values f_i' for all $i = 1,...,n_1$, the smallest shared fitness value is determined $f_1^{\min}$ of all f_i' in the first non-domination level. Thereafter, the individuals in the second non-domination level are all assigned a dummy fitness equal to a number smaller than the smallest shared fitness of the previous front $f_2 = f_1^{\min} - \varepsilon_1$ where ε_1 is a small positive number. This makes sure that no solution in the second front has a shared fitness better than that of any solution in the first front. This maintains a pressure for the solutions to lead towards the Pareto-optimal region. The sharing method is again used among the individuals of second front and shared fitness of each individual is found. This procedure is continued till all individuals are assigned a shared fitness.

3-Fonseca and Fleming's Multiobjective Genetic Algorithm (FFGA)

Fonseca and Fleming (1993)[22] proposed a Pareto-based ranking procedure (here the acronym **FFGA** is used), where an individual's rank equals the number of solutions encoded in the population by which its corresponding decision vector is dominated as depicted in figure 7c For example, an individual x_i at generation t ,which is dominated by p_i^t individuals in the current generation. Its current position in the individuals rank can be given by:

$$rank(x_i, t) = 1 + p_i^{(t)}$$

All non-dominated individuals are assigned rank 1, while dominated ones are penalized according to the population density of the corresponding region of the trade-off surface.

Fitness assignment is performed in the following way:

1. Sort population according to rank.
2. Assign fitness to individuals by interpolating from the best (rank1) to the worst (rank $n^* \leq N$) according to some function, usually linear, but not necessarily
3. Average the fitness of individual with the same rank, so that all of them will be sampled at the same rate. This procedure keeps the global population fitness constant while maintaining appropriate selective pressure, as defined by the function used. The algorithm of Fitness Assignment in FFGA are shown in figure 6

(Fitness Assignment in FFGA)
INPUT: p_t (population)
Output: F (fitness value)
Step 1: For each individual i calculate its rank.
Step 2: sort population according to rank.
Step 3: Assign fitness to individuals by interpolating from the best (rank1) to the worst (rank $n^* \leq N$) according to some function.
Step 4: Average the fitness of individual with the same rank, so that all of them will be sampled at the same rate.

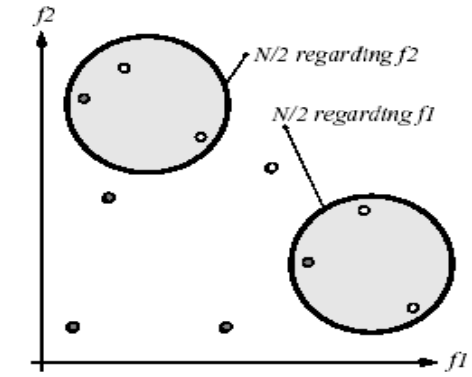
Fig. 6: Fitness Assignment in FFGA

They use a niche formulation method to distribute the population over the Pareto-optimal region, but instead of performing sharing on the parameter values, they have used sharing on *objective function* values. Fonseca and Flemming [5]gave a simple estimation of σ_{share} in the objective function space by solving the (q-1)-order polynomial equation

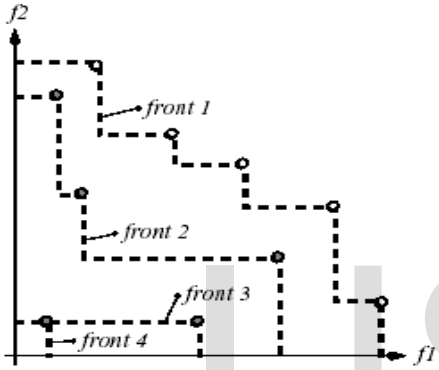
$$N\sigma_{share}^{q-1} - \frac{\prod_{i=1}^{q}(M_i - m_i + \sigma_{share}) - \prod_{i=1}^{q}(M_i - m_i)}{\sigma_{share}} = 0$$

Where q is the dimension of the objective vector , and M_i and m_i are maximum values of each objective, respectively. This maintains diversity in the objective function values but may not maintain diversity in the parameter set, which is an important issue for the decision maker.

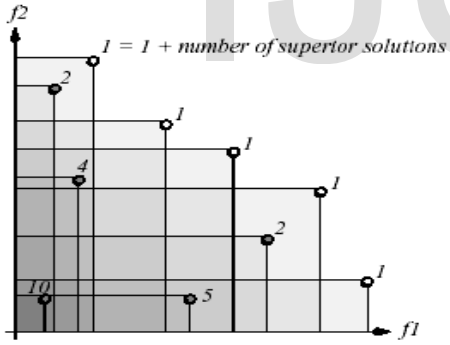
So the major difference between MOP and single objective optimization is the selection process , that is, how to parent to construct mating pool thus the flowchart of SGA are still as it for single optimization but STEP 3 will be changed such as the previous three methods or any other methods for handling MOP.



(a) VEGA



(b) NSGA



(c) FFGA

Fig. 7: Illustration of three selection mechanisms in objective space

8-APLICATIONS OF GENETIC ALGORITHMS

Since the Genetic Algorithm can be used to solve both unconstrained and constrained problems it is merely a way to obtaining a solution in a standard optimization problem. Thus it can be used to solve classic optimization problems such as maximizing volume while minimization the amount of material required to produce a container. By applying the Genetic Algorithm to linear, nonlinear programming problems and multiobjective, it is possible to solve typical problems such as the diet problem (choosing the cheapest diet from a set of foods that must meet certain nutritional requirements). Another area where Genetic Algorithms can be applied is combinatorial optimization problems including several common computer science problems such as the knapsack, traveling salesman, and job scheduling problems. In the following section several common applications where the Genetic Algorithm can be applied.

Reliability Optimization: The reliability of a system can be defined as the probability that the system has operated successfully over a specified interval of time under stated conditions. Many systems play a critical role in various operations and if they are down then the consequences can be pretty severe. Measures of reliability for systems such as communication switches is desired in order to access current reliability and also determine areas where reliability can be improved. Optimization in this field often involves in finding the best way to allocate redundant components to systems. Components are assigned probabilities to effectively gauge their reliability [27],[28],[29].

Job-Shop Scheduling: Imagine there is a sequence of machines that each performs a small task in a production line. These machines are labeled from 1 to m. For a single job to be completed work must be done first with machine 1, then machine 2, etc., all the way to machine m. There are a total of n jobs to be done and each job requires a certain amount of time on each machine (note that the amount of time required on one machine may vary from one job to another). A machine can only work on one job at any given time and once a machine starts work it cannot be interrupted until it has completed its task. The objective is to find the ideal schedule so that the total time to complete all n jobs is minimized [30],[31],[32],[33].

Transportation: The transportation Problem involves shipping a single commodity from suppliers to consumers to satisfy demand via the minimum cost. Assume that the supply equals the demand. There are m suppliers and n consumers. The cost of shipping one unit from a single supplier to each consumer is known. The problem is to find the best allocation of the commodity at the suppliers so that the demand can be satisfied and the lowest costs are incurred [34],[35],[36],[37],[38],[39].

Machine learning: GAs has been used for many machine learning tasks, including classification and prediction tasks, such as prediction of weather and protein structure. GAs have been used to evolve some particular aspects of machine learning systems, such as weights of neural networks, rules for learning classifier systems or symbolic production systems and sensors for robots[40],[41],[42].

Economics : GAs have been used to model processes of innovation, the development of bidding strategies , and the emergence of economic market[43],[44],[45].

Electrical Power Systems

Optimal power flow OPF is one of the main functions of power generation operation and control in electrical power systems. It determines the optimal setting of generating units. It is therefore of great importance to solve this problem as quickly and accurately as possible [46],[47].

9-CONCLUSION

The Genetic Algorithm is a relatively simple algorithm that can be implemented in a straightforward manner. It can be

applied to a wide variety of problems including unconstrained and constrained optimization problems, nonlinear programming, stochastic programming, and combinatorial optimization problems with single or multiple objectives. An advantage of the Genetic Algorithm is that it works well during global optimization especially with poorly behaved objective functions such as those that are discontinuous or with many local minima. It also performs adequately with computationally hard problems. Finally it can be believed that GAs may hopefully be a new effective approach for solving complex real applications.

REFERENCES

[1] Goldberg, D. E. ," Genetic Algorithms in Search, Optimization and Machine Learning". Addison Wesley Publishing company ,(1989).

[2] Holland, J.H., " Adaptation Natural and artificial systems". The University of Michigan press, AnnArbor,USA,1975.

[3] Deb, K , "An introduction to genetic algorithms", Sadhana, vol. 24, parts 4&5, August & October, pp 93-315,(1999).

[4] Gen, M. and Cheng, R., "Genetic Algorithms and Engineering Optimization". John Wily & Sons New York, (2000).

[5] Michalewicz, Z., "Genetic Algorithms + Data Structures = Evolution Programs." Springer-Verlag, 3rd Edition, (1996).

[6] Yingsong Zheng , Sumio Kiyooka , Genetic Algorithm Applications, Assignment #2 for Dr. Z. Dong ,Nov. 5, 1999, http://chawalit.siit.tu.ac.th/lib/exe/fetch.php?media=dissertations:ga_app.pdf.

[7] Ronald E. Shaffer , Practical Guide to genetic algorithm, http://www.dracica.sk/ diplom/lit/practga.html, Dec. 2013

[8] Deb, K. "Multi-objective optimization using evolutionary algorithms" NY, USA: Wiley (2001).

[9] Bazaraa M. S.," Nonlinear Programming Theory and Algorithms" New York , John Willy &Sons, (1979).

[10] Rao, S. S., "Optimization Theory And Application". Wiley Eastern Limited. New Delhi, (1991).

[11] Taha ,H. A., " Operation Reasearch : An Introduction" , 6th Ed , Collier Macmillan , London UK.(1992).

[12] Miettinen K. "Non-linear multiobjective optimization" Dordrecht: Kluwer Academic Publisher (2002).

[13] Gen, M. and Cheng, R., "Genetic Algorithms &Engineering design". John Wily & Sons New York, (1997).

[14] Kesheng Wang ,Intelligent Condition Monitoring and Diagnosis Systems: A Computational intelligence approach, IOS press, Netherlands,2003.

[15] Michalewicz, Z. and Schoenauer," Evolutionary Algorithms for Constrained Parameter Optimization Problems". Evolutionary computation 4(1)1-32,(1996).

[16] Michalewicz, Z., "A Survey of Constraint Handling Techniques In Evolutionary Computation Methods". Proceeding of the 4th Annual Conference on Evolutionary Programming, pp135-155, MIT Press, Cambridge , MA, (1995).

[17] Michalewicz, Z. and G. Nazhiyath. " Genocop III : A co-evolutionary algorithm for numerical optimization problems with non linear constraints". In D. B. Fogel (Ed.), Proceedings of the second IEEE International Conference on Evolutionary Computation, PP. 647-651. IEEE Press,(1995).

[18] Coello C., " Treating Constraints as Objectives for Single-Objective Evolutionary Optimization. Engineering Optimization", 32(3):275-308, (2000).

[19] Hiams, Y.Y, And Chankong, V.," Multiobjective Decision Making ",NEW YORK : NorthHolland,(1986).

[20] Steuer, R. E.," Multiple criteria optimization: Theory, computation, and application". New York: Wiley,(1986).

[21] Hajela, P. and Lin, C.Y. " Genetic search strategies in multicriterion optimal design" . Structural Optimization 4, 99–107,(1992).

[22] Fonseca C. M. And Fleming, P.J, "Genetic Algorithms For Multiobjective Optimization : Formulation, Discussion, And Generalization" . Proceeding Of The Fifth International Conference On Genetic Algorithms. Pp 416-423, (1993).

[23] Srinivas, N. And Deb , K. , " Multiobjective Optimization Using Nondominated Sorting In Genetic Algorithms " Evolutionary Computation ,2(3): 221-248,(1994).

[24] Zitzler, E., Deb, K., Thiele, L., Comparison of multiobjective evolutionary algorithms: Empirical results. Evolutionary Computation, 8(2): 173-195 (2000).

[25] Schaffer, J. D.," Multiple objective optimization with vector evaluated genetic algorithms". Proceedings of the First International Conference on Genetic Algorithms, 93–100,(1985).

[26] Srinivas, N. And Deb , K. , " Multiobjective Optimization Using Nondominated Sorting In Genetic Algorithms " Evolutionary Computation ,2(3): 221-248,(1994).

[27] Nikhil Gupta, Anil Swarnkar, K.R. Niazi , Distribution network reconfiguration for power quality and reliability improvement using Genetic Algorithms , International Journal of Electrical Power & Energy Systems, Volume 54, January 2014,Pages 664-671.

[28] Vijay Rathod, Om Prakash Yadav, Ajay Rathore, Rakesh Jain ,Optimizing reliability-based robust design model using multi-objective genetic algorithm , Computers & Industrial Engineering, Volume 66, Issue 2, October 2013, Pages 301-310.

[29] Laxminarayan Sahoo, Asoke Kumar Bhunia, Parmad Kumar Kapur , Genetic algorithm based multi-objective reliability optimization in interval environment ,Computers & Industrial Engineering, Volume 62, Issue 1, February 2012, Pages 152-160.

[30] Deming Lei , Co-evolutionary genetic algorithm for fuzzy flexible job shop scheduling, Applied Soft Computing, Volume 12, Issue 8, August 2012, Pages 2237-2245.

[31] Ren Qing-dao-er-ji, Yuping Wang, Xiaoli Wang , Inventory based two-objective job shop scheduling model and its hybrid genetic algorithm, Applied Soft Computing, Volume 13, Issue 3, March 2013, Pages 1400-1406.

[32] James C. Chen, Cheng-Chun Wu, Chia-Wen Chen, Kou-Huang Chen , Flexible job shop scheduling with parallel machines using Genetic Algorithm and Grouping Genetic Algorithm, Expert Systems with Applications, Volume 39, Issue 11, 1 September 2012, Pages 10016-10021.

[33] Wannaporn Teekeng, Arit Thammano , Modified Genetic Algorithm for Flexible Job-Shop Scheduling Problems, Procedia Computer Science, Volume 12, 2012, Pages 122-128.

[34] M.M. Lotfi, R. Tavakkoli-Moghaddam , A genetic algorithm using priority-based encoding with new operators for fixed charge transportation problems , Applied Soft Computing, Volume 13, Issue 5, May 2013, Pages 2711-2726.

[35] Fanrong Xie, Renan Jia, Nonlinear fixed charge transportation problem by minimum cost flow-basedgenetic algorithm, Computers & Industrial Engineering, Volume 63, Issue 4, December 2012, Pages 763-778.

[36] J. Behnamian, S.M.T. Fatemi Ghomi, F. Jolai, O. Amirtaheri , Minimizing makespan on a three-machine flowshop batch scheduling problem with-transportation using genetic algorithm , Applied Soft Computing, Volume 12, Issue 2, February 2012, Pages 768-777.

[37] A.A. Mousa " I-GA: Improved Genetic Algorithm for Multiobjective Transportation Problem", 18th international conference on computer theory and applications, ICCTA 2008,11-13October 2008, pp 169-176, Alexandria , Egypt.

[38] A.A. Mousa, (2010) 'Using genetic algorithm and TOPSIS technique for multi-objective transportation problem: a hybrid approach', International Journal of

Computer Mathematics, 87: 13, 3017 -3029, First published on: 29 June 2010 (iFirst).

[39] A. A. Mousa, hamdy M. Geneedy and Adel Y. elmekawy, efficient evolutionary algorithm for solving multiobjective transportation problem, Journal of Natural Sciences and Mathematics, Qassim University, Vol. 4, No 1, pp 79-96, June 2010.

[40] Xiao-Wei Xue, Min Yao, Zhaohui Wu, Jianhua Yang , Genetic Ensemble of Extreme Learning Machine, Neurocomputing, In Press, Accepted Manuscript, Available online 16 November 2013.

[41] Jiadong Yang, Hua Xu, Peifa Jia , Effective search for genetic-based machine learning systems via estimation of distribution algorithms and embedded feature reduction techniques ,Neurocomputing, Volume 113, 3 August 2013, Pages 105-121.

[42] Hongming Yang, Jun Yi, Junhua Zhao, ZhaoYang Dong ,Extreme learning machine based genetic algorithm and its application in power system economic dispatch Neurocomputing, Volume 102, 15 February 2013, Pages 154-162.

[43] QiSen Cai, Defu Zhang, Bo Wu, Stehpen C.H. Leung , A Novel Stock Forecasting Model based on Fuzzy Time Series and GeneticAlgorithm , Procedia Computer Science, Volume 18, 2013, Pages 1155-1162.

[44] He Ni, Yongqiao Wang , Stock index tracking by Pareto efficient genetic algorithm, Applied Soft Computing, Volume 13, Issue 12, December 2013, Pages 4519-4535.

[45] Janko Straßburg, Christian Gonzàlez-Martel, Vassil Alexandrov, Parallel genetic algorithms for stock market trading rules Procedia Computer Science, Volume 9, 2012, Pages 1306-1313.

[46] M.S.Osman , M.A.Abo-Sinna , and A.A. Mousa " A Solution to the Optimal Power Flow Using Genetic Algorithm "Journal of Applied Mathematics & Computation (AMC) vol 155, No. 2, 6 August (2004) pp 391-405. (Top 25 Hottest Articles, July. to Sept. 2005), from M. Sc. Dissertation.

[47] Osman M.S., M.A.Abo-Sinna, and A.A. Mousa "Epsilon-Dominance based Multiobjective Genetic Algorithm for Economic Emission Load Dispatch Optimization Problem, Electric Power Systems Research 79 (2009) 1561–1567.