FINDING FAILED ELEMENT POSITIONS IN LINEAR ANTENNA ARRAYS USING GENETIC ALGORITHM

Ali Akdağlı Kerim Güney Dervis Karaboğa* Bilal Babayiğit*
e-mail: akdagli@erciyes.edu.tr e-mail: kguney@erciyes.edu.tr e-mail:karaboga@erciyes.edu.tr
Erciyes University, Faculty of Engineering, Department of Electronics Engineering, 38039, Kayseri, Turkey
**Erciyes University, Faculty of Engineering, Department of Computer Engineering, 38039, Kayseri, Turkey*

Key words: Antenna array, failed elements, genetic algorithm

ABSTRACT
In this work, a simple approach based on the genetic algorithm (GA) is presented for finding the positions of defective elements in a linear antenna array. The GA is used to minimize the difference between the samples of far-field power pattern of an array with failed elements and the measured one. Simulation results are shown for 30 elements Chebyshev array with one and multiple failed elements to illustrate the performance of the presented approach.

I. INTRODUCTION
Element failures in antenna arrays destroy the symmetry and cause unacceptable pattern distortion, for example, a significant increase of both sidelobe and ripple level of the power pattern. This problem can be solved by replacing the defective elements in aircraft antennas. However, this is a critical problem in space platforms. Active array antennas have the advantage that the radiation pattern can be restored by changing their feeding distribution from base station [1]. In order to compensate the effects of failed elements, the excitations of unfailed elements can be readjusted, thus a pattern with a minimum loss of quality with respect to the original one is produced. In the literature, several methods [2-5] have been proposed to perform this compensation by numerically finding a new set of excitations of the unfailed elements that optimizes some objective function. However, it is obvious that these methods need to know the position and the number of the failed elements in the array.

Recently, Rodriguez at al. [6] demonstrated how to use the genetic algorithm [7, 8] (GA) for finding the defective elements in a planar array. It is well known that the GA, which is based on natural selection and genetic science, is a parallel, robust, and probabilistic search technique that is simple and easily implemented without gradient calculation, compared to the conventional gradient-based search procedures. Most important of all, the GA also provides a mechanism for global search that is not easily trapped in local minima. Due to the fascinating features mentioned above, the GA has been applied to a wide variety of electromagnetic design and antenna problems. In this study, we used the GA to find the number and positions of defective elements in a linear antenna array. A cost function which represents the difference between the some samples of damaged radiation pattern and measured one is constructed and minimized by means of the GA.

The GA version adopted here is different from the standard GA versions existing in the literature since it uses an adaptive mutation rate strategy. In the following section, a basic GA and the GA used in this work are described.

II. GENETIC ALGORITHM
The GA is a general purpose optimization algorithm with a probabilistic component that provides a means to search poorly understood, irregular spaces. John Holland [7] originally developed the GA and provided its theoretical foundation in his book. Holland developed the GA to simulate some of the processes observed in natural evolution. Evolution is a process that operates on chromosomes rather than on living beings. Natural selection links chromosomes with the performance of their decoded structure. The process of natural selection cause those chromosomes that encode successful structures to reproduce more often than those that do not. Recombination process creates different chromosomes in children by combing material from the chromosomes of the two parents. Mutation may cause the chromosomes of children to be different from those of their parents.

The goal of the GA is to find a set of parameters that minimizes the output of a function. The GA differs from most optimization methods, because they have the following characteristics:

1. It works with a coding of the parameters, not the parameters themselves.

2. It searches from many points instead of a single point.
3. It doesn't use derivatives.
4. It uses random transition rules, not deterministic rules.

The GA is a stochastic optimization algorithm which is as robust as it is versatile. The algorithm repetitively applies three stochastic operators on a population of designs, reproduction, crossover, and mutation. The GA has the desirable attribute that by operating on a large population of designs, and by implementing stochastic transition rules, it avoids getting trapped into a local minima. Another attractive feature of this algorithm is that it can generate a number of distinct and high performance designs and offers the designer a number of candidates for the final design.

The GA requires the problem of maximization or minimization to be stated in the form of a cost (objective) function. In a GA, a set of variables for a given problem is encoded into a string (or other coding structure), analogous, to a chromosome in nature. Each string, therefore, contains a possible solution to the problem. To determine how well a chromosome solves the problem, it is first broken down into the individual sub strings which represent each variable and these values are then used to evaluate the cost function, yielding a 'fitness'. The GA selects parents from a pool of strings (population) according to the basic criteria of 'survival of the fittest'. It creates new strings by recombining parts of the selected parents in a random manner. In this manner, the GA is able to use historical information has a guide through the search space.

The repopulating of the next generation is done using three methods: reproduction, crossover, and mutation. Through reproduction, strings with high fitnesses receive multiple copies in the next generation while strings with low fitnesses receive fewer copies or even none at all. Crossover refers to taking a string, splitting with into two parts a randomly generated crossover point and recombining it with another string which has also been split at the same crossover. This procedure serves to promote change in the best strings which could give them even higher fitnesses. Mutation is the random alteration of a bit in the string which assists in keeping diversity in the population.

The GA works through function evaluation, not through differentiation or other such means. Because of this trait, a GA dose not care what type of problem it is asked to minimize only that it be properly coded. Thus the GA is able to solve a wide range of problems: linear, nonlinear, discontinuous, discrete, etc.

A Basic Genetic Algorithm

The pseudo code of a basic GA is as below:

```
        Initialize P(t=0); /*P(0)=initial population*/
        Evaluate each member in P(t);
        While(not termination condition) do
{
        Generate P(t+1) from P(t) as follows:
                {
                Select individuals from P(t) on basis of fitness (selection operation);
                Recombine them using genetic operators (crossover and mutation operations);
                }
        t=t+1
        Evaluate each member in P(t);
}
```

The initial population required at the start of the algorithm is a set of number strings generated by the random number generator. Each string is a representation of a solution to the optimization problem being addressed. Binary strings are commonly employed. Associated with each string is a fitness value as computed by the evaluation unit. A fitness value is a measure of the goodness of the solution that it represents. The aim of the genetic operators is to transform the set of the strings into the sets with higher fitness values.

The reproduction operator performs a natural selection function known as "seeded selection". Individual strings are copied from one set (representing a generation of solutions) to the next according to their fitness value, the greater the probability of a string being selected for the next generation.

The crossover operator chooses pairs of strings at random and produces new pairs. The simplest crossover operation is to cut the original parent strings at a randomly selected point and exchange their tails. The number of crossover operations is governed by a crossover rate.

The mutation operator randomly mutates or reverses the values of bits in a string. The number of mutation operations is determined by a mutation rate.

A phase of the algorithm consists of applying the evaluation, reproduction, crossover and mutation operations. A new generation of solutions is produced with each phase of the algorithm.

The Genetic Algorithm Used in This Work

The GA version adopted in this study was proposed by Karaboga at al. [9]. This version is different from standard GA versions existing in the literature since it uses an adaptive mutation rate strategy. The pseudo code of this algorithm is given below:

```
        Initialize P(t=0); /*P(0)=initial population*/
        Evaluate each member in P(t);
        While (not termination condition) do
{
        Generate P(t+1) from P(t) as follows:
                {
                Select individuals from P(t) using seeded selection, random selection and elite procedures;

                Recombine selected individuals applying crossover and mutation operations (mutation operator uses an adaptive mutation rate strategy, not constant mutation rate);
                }
        t=t+1
        Evaluate each member in P(t);
        Apply fitness scaling;
}
```

Note a fitness scaling unit for normalizing the fitness values computed by the evaluation unit. A scaling window is used in this normalization process to distinguish between good and better solutions. The reproduction unit also implements a random selection procedure, controlled by a parameter called the generation gap and an elite procedure for preserving the fittest solution in each generation, in addition to the seeded selection process described earlier.

III. THE METHOD

The far field array factor of an equally spaced linear array with N isotropic elements oriented along the z-axis can be written as:

$$AF\,(\theta) = \sum_{n=1}^{N} a_n\, e^{jn\frac{2\pi}{\lambda} d \sin(\theta)} \qquad (1)$$

where a_n is the excitation of the n*th* element, d is the interelement spacing, λ is the wavelength, and θ is the angle from broadside. If n*th* element is failed, its excitation a_n is assumed to be zero. To find the number and positions of failed elements by using the GA, the following cost function is employed.

$$C = \sum_{k=1}^{M} \left| AF_o(\theta_k) - AF_m(\theta_k) \right| \qquad (2)$$

where M is the number of samples used in the comparison. $AF_o(\theta_k)$ and $AF_m(\theta_k)$ represent the obtained value of the pattern of array whose defective element positions will be determined by GA and the value of the measured pattern at the k*th* sample point, respectively. Since it is very difficult to measure whole damaged pattern in all directions, a limited number of samples M is used.

IV. SIMULATION RESULTS

In the study, to show the effectiveness of the GA, a real failed antenna array is not performed, but it is simulated. In the simulation process, a 30-dB Chebyshev array pattern for 30 equispaced elements with $\lambda/2$ interelement spacing is used as the initial pattern. The measured data are produced by randomly cancelling the some element excitations of this Chebyshev array. For simplicity, the measurement errors are ignored in this work. In the optimization process, the values of GA parameters such as population size, crossover rate and mutation rate are chosen as 30, 0.8 and 0.02, respectively. Since there are two possibilities for the status of each array element (failed or unfailed), one bit (0 for failed and 1 for unfailed element) satisfies encoding of each array element. Thus, 30 bits are used for 30-elements initial Chebyshev array. The calculations are performed on a personal computer with a Pentium Celeron processor running at 1700 MHz.

To verify the capability of the GA for finding the defective elements, a number of element failure cases are investigated. The number of samples (M) is 5 and this is sufficient to find the accurate solution. The sample directions are chosen as 36^o, 72^o, 108^o, 144^o and 180^o. The simulation results using 10 random configuration of one, two and three defective elements are obtained and listed in Tables 1-3, respectively. From Tables 1-3, it is seen that the GA can find the failed elements in all cases.

Table 1. Simulation results obtained by using 10 random configurations of one defective element

Failed element position	Elapsed time (s)
1	1
3	1
6	2
10	1
12	2
15	1
17	1
21	1
25	1
29	2

It is noted that the elapsed times given in Tables 1-3 do not depend on the positions of the failed elements. The reason of different elapsed time values for the same problem (for example one failed element) is that the GA produces the initial solutions randomly. Hence, even for the same problem, the elapsed time might change for each

Table 2. Simulation results obtained by using 10 random configurations of two defective elements

Failed element positions	Elapsed time (s)
1,30	2
2, 17	3
7, 25	7
9, 15	3
3, 25	2
13, 16	2
10, 21	2
18, 20	5
23, 29	2
5, 26	10

Table 3. Simulation results obtained by using 10 random configurations of three defective elements

Failed element positions	Elapsed time (s)
1, 9, 20	32
5, 26, 30	120
2, 8, 25	222
6, 12, 22	36
17, 23, 29	70
8, 12, 16	13
3, 6, 15	200
11, 18, 27	18
9, 13, 22	12
17, 24, 28	44

run. However, it is obvious from Tables 1-3 that the elapsed time increases as the number of failed elements increases.

It should be also noted that when the number of elements in the array is increased or if measurement errors are included, the problem of finding defective elements gets more difficult. In this case, the number of measured sample required might increase and the GA can still achive the sufficient results but it may take much more computational time.

V. CONCLUSION

The failed element positions in a linear antenna array are determined with the use of the GA. The measured data for the damaged pattern are produced by simulating a 30-elements Chebyshev array with some defective elements instead of a real failed antenna array. One, two and three defective elements case are investigated. Simulation results obtained by using only 5 samples of damaged pattern show that the GA is capable of finding the right configuration in all cases.

REFERENCES

1. R. J. Mailloux, Phased Array Antennas Hanbook, Artech House, Inc., Norwood, MA, 1994.
2. B. Yeo, Y. Lu, Array Failure Correction with a Genetic Algorithm, IEEE Transactions on Antennas and Propagation, vol. 47, pp. 823-828, 1999.
3. T. J. Peters, A Conjugate Gradient-Based Algorithm to Minimize the Sidelobe Level of Planar Arrays with Element Failures, IEEE Transactions on Antennas and Propagation, vol. 39, pp. 1497-1504, 1991.
4. R. J. Mailloux, Array Failure Correction with a Digitally Beamformed Array, IEEE Transactions on Antennas and Propagation, vol. 44, pp. 1542-1550, 1996.
5. J. A. Rodriguez, F. Ares, E. Moreno, G. Franceschetti, Genetic Algorithm Prosedure for Linear Array Failure Correction, Electronics Letters, vol. 36, pp. 196-198, 2000.
6. J. A. Rodriguez, F. Ares, H. Palacios, J. Vassal'lo, Finding Defective Elements in Planar Arrays Using Genetic Algorithm, Progress in Electromagnetics Research, PIER 29, pp. 25-37, 2000.
7. J. H. Holland, Adaptation in Natural and Artificial System, Ann Arbor: University of Michigan Press, USA, 1975.
8. L. Davis, Handbook of Genetic Algorithms, Van Nostrand Reinhold, NY, New York, 1991.
9. D. Karaboga, K. Güney, N. Karaboga, Simple and Accurate Effective Side Length Expression Obtained by Using a Modified Genetic Algorithm for the Resonant Frequency of an Equilateral Triangular Microstrip Antenna, International Journal of Electronics, vol. 83, pp. 99-108, 1997.