Abstract—Large arrays are difficult to thin in order to obtain low sidelobes. Traditional statistical methods of aperiodic array synthesis fail far short of optimum configurations. Traditional optimization methods are not well suited for optimizing a large number of parameters or discrete parameters. This paper presents how to optimally thin an array using genetic algorithms. The genetic algorithm determines which elements are turned off in a periodic array to yield the lowest maximum relative sidelobe level. Simulation results for 200 element linear arrays and 200 element planar arrays are shown. The arrays are thinned to obtain sidelobe levels of less than -20 dB. The linear arrays are also optimized over both scan angle and bandwidth.

I. INTRODUCTION

A PERIODIC arrays generate a low sidelobe amplitude taper by strategically positioning equally weighted elements. Simple analytical methods for deriving the element positions to obtain a desired sidelobe level are not available [1]. Instead, most aperiodic array synthesis methods try to relate the element density at a region of the array to the amplitude density of a low sidelobe amplitude taper for the same size aperture [2]. The element density is greatest at the center of the array and gradually decreases toward the edges. In general, sidelobes close to the main beam decrease, while those far from the main beam increase [3] (which is usually quite acceptable). Aperiodic array synthesis methods either target a rms sidelobe level or a maximum relative sidelobe level with a given probability [4].

Thinning an array means turning off some elements in a uniformly spaced or periodic array to create a desired amplitude density across the aperture. An element connected to the feed network is "on", and an element connected to a matched or dummy load is "off". Thinning an array to produce low sidelobes is much simpler than the more general problem of nonuniform spacing the elements. Nonuniform spacing has an infinite number of possibilities for placement of the elements. Thinning has \(2^Q\) possible combinations, where \(Q\) is the number of array elements. If the array is symmetric, then the number of possibilities is substantially smaller. Thinning may also be thought of as a quantized amplitude taper where the amplitude at each element is represented by one bit.

Thinning a large array for low sidelobes involves checking a rather large number of possibilities in order to find the best thinned aperture. Exhaustive checking of all possible element combinations is only practical for small arrays [5]. Most optimization methods (including down-hill simplex, Powell’s method, and conjugate gradient) are not well suited for thinning arrays. They can only optimize a few continuous variables and get stuck in local minima [6]. Also, these methods were developed for continuous parameters, whereas the array thinning problem involves discrete parameters. Dynamic programming can optimize a large parameter set (many elements), but it is vulnerable to local minima [11].

Simulated annealing and genetic algorithms [7] are optimization methods that are well suited for thinning arrays. There is no limit to the number of variables that can be optimized. Although quite slow, these algorithms can handle very large arrays. These methods are global in that they have random components that test for solutions outside the current minimum, while the algorithm converges. The global nature of the algorithms and the lack of derivative information causes them to converge very slowly compared to other nonglobal methods.

Ruf used simulated annealing to optimize low-redundancy linear arrays [8]. These antennas are used in radio astronomy to estimate scene radiance. Astronomers are interested in designing arrays that properly sample the spatial distribution they observe. His approach optimized much larger linear array configurations than ever before.

The purpose of this paper is to find a thinned array that produces the lowest maximum relative sidelobe level (rsll). A genetic algorithm is used to numerically optimize both linear and planar arrays [9]. Genetic algorithms are modeled after genetic recombination and evolution. The algorithms encode parameters in binary strings called genes and perform the genetic operations of reproduction, crossover, natural selection, and mutation to arrive at the optimum solution. These algorithms arrive at better thinning configurations for arrays than previous optimization attempts or statistical attempts. Other optimization methods cannot be applied to large arrays, while statistical methods cannot find optimum solutions. Excellent results are obtained using genetic algorithms to optimize linear arrays and planar arrays with 200 elements. The linear arrays can be optimized for maximum relative sidelobe levels less than -20 dB over a range of scan angles and frequencies.

II. THE GENETIC ALGORITHM

The goal of the genetic algorithm is to find a set of parameters that minimizes the output of a function. Genetic algorithms differ from most optimization methods, because they have the following characteristics [10].

1) They work with a coding of the parameters, not the parameters themselves.
2) They search from many points instead of a single point.
3) They don’t use derivatives.
4) They use random transition rules, not deterministic rules.
Fig. 1 is a flow chart of a genetic algorithm. The following paragraphs describe the details of each step in the algorithm. Steps are labeled as A through F for easy reference.

Values for all the parameters are represented by a binary code (step A). Each encoded parameter is placed side by side to form a long binary string called a gene. Every gene has an associated output corresponding to the function evaluated at the quantized parameters. Thus, the genetic algorithm has a finite, but possibly very large, number of parameter combinations to check. A gene with $N; B$-bit parameters has a total of $2^{NB}$ possible genes. If the parameters are continuous, then the genetic algorithm limits performance due to quantization errors associated with the binary encoding of the parameters. On the positive side, genetic algorithms are ideally suited for optimization of functions with discrete parameters.

A thinned array has discrete parameters. One bit represents the element state as "on" = 1 or "off" = 0. For example, a six element array may be represented by 101101, where elements 2 and 5 are turned "off." Assuming the linear array is symmetric about its center allows the $2N$ element array to be represented by a gene with $N$ bits. Our six-element array example can then be represented by the gene 101. The fitness associated with this gene is the maximum relative sidelobe level (rsll) of its associated far-field pattern. The function in this paper is the relative far-field pattern of an array of point sources. Its output to be minimized is the maximum rsll. The parameters affecting the output are whether an antenna element is on or off.

Genetic algorithms model genetic recombination and evolution in nature [10]. As in nature, the gene is the basic building block. Genetic algorithms start with a random sampling of the output space. Many of the genes from this list have terrible maximum rsll's. Genes that produce a superior output survive, while genes that produce a weak output die off. For instance, consider the list of eight randomly generated genes (step B) in Table I.

These genes represent a 20-element array of isotropic point sources spaced $\frac{\lambda}{2}$ apart. Each gene has an associated rsll calculated from

$$FF(u) = \max \left[ \sum_{n=1}^{N} a_n \cos(2\pi n du + \delta_n) \right] \frac{elpat(n)}{FF_{max}}$$

where

- $2N$ = number of elements in the array.
- $a_n$ = amplitude weight of element $n$ = \begin{cases} 1 & \text{off} \\ 0 & \text{on} \end{cases}
- $d$ = spacing between elements.
- $u = \cos(\phi)$.
- $\phi$ = angle measured from line passing through antenna elements.
- $\delta_n$ = steering phase = $-2\pi n d u_s$.
- $elpat(u)$ = element pattern.
- $c_o$ = constant.
- $FF_{max}$ = peak of main beam.

The region of $u$ for which $FF$ is valid excludes the main beam. The first null for a uniform array occurs at $u = \frac{1}{2Nd}$. Thinning an array causes the null to move, so a constant, $c_o$, is needed in the formula to adjust for the first null location.

Which of these output values from Table I are fit enough to survive? First, let's rank them from best to worst, as shown in Table II (step C). The most common suitability criterion is to discard (step D) the bottom half and keep the top half of the list. Note, however, that only gene 2 has a rsll of less than $-13$ dB. Another possible criterion is to discard any gene that has a rsll of less than $-13$ dB. This limitation speeds convergence of the algorithm. Since a fully populated array has a rsll of slightly less than $-13$ dB, only one of these genes shows any improvement over a nonthinned array.

Table I

<table>
<thead>
<tr>
<th>Gene</th>
<th>Code</th>
<th>Max rsll (dB)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1111011001</td>
<td>-9.06</td>
</tr>
<tr>
<td>2</td>
<td>1111111001</td>
<td>-14.08</td>
</tr>
<tr>
<td>3</td>
<td>1110111110</td>
<td>-12.13</td>
</tr>
<tr>
<td>4</td>
<td>1101111101</td>
<td>-9.71</td>
</tr>
<tr>
<td>5</td>
<td>1111111101</td>
<td>-11.09</td>
</tr>
<tr>
<td>6</td>
<td>1110110100</td>
<td>-7.31</td>
</tr>
<tr>
<td>7</td>
<td>1111110010</td>
<td>-12.76</td>
</tr>
<tr>
<td>8</td>
<td>1011110101</td>
<td>-8.70</td>
</tr>
</tbody>
</table>

*Fig. 1. Flow chart of a genetic algorithm.*
TABLE II
GENES RANKED ACCORDING TO FITNESS

<table>
<thead>
<tr>
<th>Gene</th>
<th>Code</th>
<th>Max rsll (dB)</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>111111001</td>
<td>-14.08</td>
</tr>
<tr>
<td>7</td>
<td>1111101110</td>
<td>-12.76</td>
</tr>
<tr>
<td>3</td>
<td>1111011110</td>
<td>-12.13</td>
</tr>
<tr>
<td>5</td>
<td>1111011101</td>
<td>-11.09</td>
</tr>
<tr>
<td>4</td>
<td>1101111001</td>
<td>-9.71</td>
</tr>
<tr>
<td>1</td>
<td>1111011001</td>
<td>-9.06</td>
</tr>
<tr>
<td>8</td>
<td>1011110101</td>
<td>-8.70</td>
</tr>
<tr>
<td>6</td>
<td>1110110010</td>
<td>-7.31</td>
</tr>
</tbody>
</table>

Fig. 2. Two genes mating. A random crossover point is selected for two parent genes. Offspring are created by swapping the bits to the right of the crossover point.

The surviving genes. Once paired, their offspring consist of genetic material from both parents. Fig. 2 shows gene 2 mating with gene 7 to produce genes 9 and 10. The random crossover point is between bits selected for each pair of genes. One of the offspring consists of the bit sequence from the left side of the crossover point of parent A and the bit sequence from the right side of the crossover point of parent B. Likewise, the other offspring consists of the bit sequence from the left side of the crossover point of parent B and the bit sequence from the right side of the crossover point of parent A. The top four genes in Table III survived natural selection and became parents of the bottom four genes. Genes 9 and 10 are offspring of genes 2 and 7, and genes 11 and 12 are offspring of genes 3 and 5.

One last step is to introduce a random mutation in the list of genes (step F). A mutation changes a zero to a one or a one to a zero. In this example only one bit in all the genes is mutated in each iteration. The mutation helps the algorithm avoid a local minimum.

The process starts over again with the parents and the offspring (step C). Table IV shows the parents and offspring (with a mutation) of Table III listed in order from best to worst. Note that bit 6 in gene 10 mutated in Table IV. Gene 10 had an output of -11.73 dB until its sixth bit mutated. The mutation caused the output to drop to -9.43 dB. Mutation usually doesn’t improve the solution. It is a very necessary part of genetic algorithms, though. Without it, genetic algorithms are more likely to get stuck in a local minimum. Natural selection, mating, and mutation will take place with these genes. The algorithm continues this process until a suitable stopping point is reached. Eventually, all the genes will be identical except for the single mutated gene.

The person programming the genetic algorithm has many variables to control and trade-offs to consider. For instance:

1) Number of bits that represent a parameter. More bits give greater accuracy but slow convergence. This is not an issue with array thinning, since each element is

TABLE III

<table>
<thead>
<tr>
<th>Gene</th>
<th>Code</th>
<th>Max rsll (dB)</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>111111001</td>
<td>-14.08</td>
</tr>
<tr>
<td>7</td>
<td>1111101110</td>
<td>-12.76</td>
</tr>
<tr>
<td>3</td>
<td>1111011110</td>
<td>-12.13</td>
</tr>
<tr>
<td>5</td>
<td>1111011101</td>
<td>-11.09</td>
</tr>
<tr>
<td>4</td>
<td>1101111001</td>
<td>-9.71</td>
</tr>
<tr>
<td>1</td>
<td>1111011001</td>
<td>-9.06</td>
</tr>
<tr>
<td>8</td>
<td>1011110101</td>
<td>-8.70</td>
</tr>
<tr>
<td>6</td>
<td>1110110010</td>
<td>-7.31</td>
</tr>
</tbody>
</table>

TABLE IV
THESE ARE THE PARENTS AND OFFSPRING RANKED AFTER MUTATION OF BIT 6 IN GENE 10

<table>
<thead>
<tr>
<th>Gene</th>
<th>Code</th>
<th>Max rsll (dB)</th>
</tr>
</thead>
<tbody>
<tr>
<td>9</td>
<td>111111010</td>
<td>-14.66</td>
</tr>
<tr>
<td>2</td>
<td>111111100</td>
<td>-14.08</td>
</tr>
<tr>
<td>7</td>
<td>111110010</td>
<td>-12.76</td>
</tr>
<tr>
<td>3</td>
<td>111011110</td>
<td>-12.13</td>
</tr>
<tr>
<td>11</td>
<td>111011101</td>
<td>-12.01</td>
</tr>
<tr>
<td>12</td>
<td>111011100</td>
<td>-11.89</td>
</tr>
<tr>
<td>5</td>
<td>111011101</td>
<td>-11.09</td>
</tr>
<tr>
<td>10</td>
<td>111100001</td>
<td>-9.43</td>
</tr>
</tbody>
</table>
represented by one bit. A continuous parameter, such as an element amplitude weight, however, is sensitive to quantization.

2) **Number of genes in the initial random population.** More genes provide a better sampling of the solution space but slow convergence.

3) **Generating the random list.** The type of probability distribution and weighting of the parameters has a significant impact on the convergence time.

4) **Natural selection.** Several methods are available for deciding which genes to discard.

5) **Pairing the genes for mating.** Genes may be paired from top to the bottom of the list, randomly, best with worst, etc.

6) **Number of mutations.** Mutations guard against the algorithm getting stuck in a local minimum but slow convergence.

7) **When does the algorithm converge?** Determining a stopping point for the algorithm is difficult. Eventually the natural selection and mating processes will cause all the genes to be the same, except for the mutated gene.

The above tradeoffs will be referenced by Roman numerals in the next section. All these considerations and more make genetic algorithms an interesting numerical method to explore. In general, I found the genetic algorithm converged well for a variety of options mentioned in the tradeoffs. The main impact of the options is on convergence speed.

### III. Optimally Thinned Arrays

The previous section gave a step-by-step description of how the algorithm worked for a small array. Applying genetic programming to optimizing small arrays is overkill, because small arrays can be optimized by looking at every possible thinning combination and selecting the best one. This section looks at more realistic applications of genetic algorithms to array thinning. Results for optimizing the rssl of a 200-element linear array and a 200-element planar array are presented. It is impractical to optimize these arrays by examining every possible combination of thinning.

Fig. 3(a) is the far-field pattern of a thinned array of 200 isotropic point sources. The element status (on or off) for half the array is shown at the top of the picture. The top left 1 is the first element to the right of array center, and the bottom right 1 is at the right edge. The left side of the array is the mirror image of the right. In this case a 77% filled array produced a maximum rssl of -22.09 dB. The aperture size is approximately 100λ, and the gain is approximately 21.9 dB (gain=\(N_tG_e\) where \(N_t\) = number of elements turned on and \(G_e\) = the element gain [11]). Only genes with a maximum rssl of -13 dB or better were allowed to survive (see 4 above). The initial list of 200 random genes (see 2) was weighted so that the center elements were more likely to be turned on than the elements towards the edge (see 3). This weighting significantly speeds convergence of the algorithm. Several sidelobes distributed throughout the far-field pattern nearly touch the -22.09 dB maximum rssl. Genes were paired from best to worst (see 5), and two mutations occurred every iteration (see 6). I stopped the algorithm after thirty iterations (see 7). Statistical aperiodic array theory predicts that over 1000 elements are needed to obtain a -22-dB maximum rssl with 90% probability for a 100λ aperture [1]. The optimum array designed here has low sidelobes and high gain, and is practical to build.

Fig. 3(b) is the far-field pattern of a thinned array of 200 sources having a \(\sin\theta\) element pattern. In this case, a 75% filled array produced a maximum rssl of -23.69 dB. Optimizing on the array with an element pattern produced results 1.6 dB better than the isotropic element array. The thinning for these two examples are quite different even though both are 200-element arrays.

Thinning the array of directional elements yields better results at broadside than thinning an array of isotropic point sources with a \(\sin\theta\) element pattern.
sources of the same size. However, steering the beam away from broadside causes a significant degradation in the maximum rsl for the array of directional elements. Fig. 4 shows the antenna pattern in Fig. 3(b) steered to 120°. The maximum rsl deteriorates to −18.8 dB. The thinned array of isotropic point sources suffers no sidelobe degradation when the main beam is steered. Sidelobe degradation for the array of directional elements may be prevented by optimizing the pattern over the range of steering angles of the array. For instance, assume the 200-element array of directional elements steers its beam between 60° and 120°. The function to be minimized now returns the maximum rsl over the range of steering angles. Excellent results are obtained by evaluating the function at only 120°. The maximum steering angle is the point of maximum attenuation of the main beam by the element pattern. Thus, the maximum rsl of all other steering angles is lower than at 120°. Fig. 5(a) is the far-field pattern of the 78% filled 200-element array of directional sources at broadside, and Fig. 5(b) is the pattern when steered to 120°. The maximum rsl over the scan limit is −22.22 dB. This maximum rsl is almost the same as that obtained for the array of isotropic point sources but is less than that obtained in Fig. 3(b).

Another possible genetic algorithm optimization problem is nulling interference. As an example, consider a 200-element array with \( d = \frac{\lambda}{2} \), a 10% bandwidth, scanning ±30°, and a known interference between \( u = 0.025 \) and 0.075. The problem is to thin the array to obtain the lowest possible sidelobe levels over the angular extent of the interference and over the bandwidth and steering angles of the array. Optimizing with the genetic algorithm produced an array that is 83% filled with a maximum rsl of −26.8 dB between \( u = 0.025 \) and 0.075 over a 10% bandwidth. Fig. 6(a) shows the far-field pattern when \( \phi = 90° \) and \( d = 0.5 \lambda \) for the thinning configuration indicated at the top of the graph. Fig. 6(b) shows the main beam steered to 120° and \( d = 0.55 \lambda \) for the same optimized thinned array. The maximum rsl of all the patterns is less than −16 dB.

The next example is thinning a 20 x 10 element planar array of isotropic point sources in a square lattice. The equation for the planar array far field pattern is given by

\[
FF(\theta, \phi) = 4 \sum_{n=1}^{N} \sum_{m=1}^{M} a_{mn} \cos[(2m-1)\pi d_y \sin \theta \cos \phi] \times \cos[(2n-1)\pi d_x \sin \theta \sin \phi]
\]

where

- \( M \) = number of elements in the \( y \)-direction,
- \( N \) = number of elements in the \( x \)-direction,
- \( d_y \) = spacing in the \( y \)-direction,
- \( d_x \) = spacing in the \( x \)-direction.

This equation assumes the array lies in the \( x-y \) plane and has symmetry about the \( x \)-axis and the \( y \)-axis. Some modifications
to the genetic algorithm for linear arrays are necessary. First, the gene may be represented in several ways. Here, each array thinning configuration was loaded into a single one-dimensional array, and the algorithm performed the optimization in the same manner as it did for the linear array. Another possibility is to load the thinning configuration into a two dimensional matrix. A matrix offers some interesting new possibilities for mating. Since the genes are two-dimensional, the crossover point becomes a crossover line. This crossover line can be drawn horizontally, vertically, or diagonally. Paired genes would swap genetic material on one side of the line. A second option is how the fitness is calculated. In the following example the fitness was the sum of the maximum sll in the $\phi = 0^\circ$ and $\phi = 90^\circ$ planes. Fig. 7 shows the elements that are turned on (white) and turned off (black). There are 108 elements turned on, so the array is 54% filled. Fig. 8 is the far-field pattern, and Fig. 9 far field pattern cuts in the $\phi = 0^\circ$ and $\phi = 90^\circ$ planes.

Other fitness criteria are possible. If the fitness is just the maximum sll in the $\phi = 0^\circ$ plane, then the optimum maximum sll is $-22.17$ dB. The maximum sll in the $\phi = 0^\circ$ plane is $-12.93$ dB. In this case, the array is 58% filled. The fitness criteria depends upon the design specifications for the array.

Another way to thin a planar array is to assume each row in the thinned array is the same and each column is the same. The $a_{mn}$ in (2) equals $a_{m}a_{n}$, where $a_{m}$ is the amplitude taper in the $y$-direction and $a_{n}$ is the amplitude taper in the $x$-direction. An optimized $40 \times 40$ planar array of isotropic point sources in a square lattice is 81% filled with a maximum sll of $-17.2$ dB. All elements in rows and columns 4, 7, 35, and 38 are turned off.

IV. CONCLUSIONS

This paper introduced the use of genetic algorithms for thinning linear and planar arrays to obtain the lowest possible
I HAUS THINNED ARRAYS USING GENETIC ALGORITHMS

phi=0, maxsll=-20.07 phi=90, maxsll=-19.76

Fig. 9. Far-field pattern cuts at $\phi = 0^\circ$ and $\phi = 90^\circ$. The maximum sidelobe level is $-20.07$ dB in the $\phi = 0^\circ$ plane and $-19.76$ dB in the $\phi = 90^\circ$ plane.

maximum sidelobe. The beauty of the genetic algorithm is that it can optimize a large number of discrete parameters. Previous methods of array thinning used statistical methods of representing an amplitude taper and fail to produce an optimum thinning. The genetic algorithm intelligently searches for the best thinning that produces low sidelobes.

This paper demonstrated several ways to apply genetic algorithms to optimize array patterns. Many additional extensions are possible, including thinning circular arrays, planar arrays with directional elements, scanning planar arrays, etc.

Genetic algorithms are slow and not useful for real-time pattern control such as adaptive nulling. On the other hand, the algorithms are quite useful for optimizing array designs. The applications in this paper concern thinning, but a wide variety of other applications in scattering and antenna theory are possible.

REFERENCES


Randy L. Haupt (M'82-S'84-M'87-SM'90) received the B.S. degree from the US Air Force (USAF) Academy in 1978, the M.S. degree from Northeastern University in 1983, and the Ph.D. degree from the University of Michigan in 1987, all in electrical engineering. He also received the M.S. degree in engineering management from Western New England College in 1981. He is currently a lieutenant colonel in the USAF and an Associate Professor at the USAF Academy. He previously worked as an engineer on the USAF Over-the-Horizon Radar system and as an antenna engineer for Rome Air Development Center. His research interests include education, antennas, radar cross section, numerical methods, and chaos theory.

Dr. Haupt is a registered Professional Engineer in Colorado and a member of Tau Beta Pi. He was named the 1993 Federal Engineer of the Year by the National Society of Professional Engineers.