MITSUBISHI ELECTRIC RESEARCH LABORATORIES
http://www.merl.com

Very Fast Subarray Position Calculation for Minimizing Sidelobes in Sparse Linear Phased Arrays

Christopher Lee, Darren Leigh, Kathy Ryall, Hiroaki Miyashita, Kazufumi Hirata

TR2006-022 November 2006

Abstract

We have developed a set of algorithms and software tools to help an antenna designer interactively explore the space of possible subarray positions in a sparse linear phase array, for the purpose of designing antennas that minimize the maximum sidelobe level of the array response. This paper presents two new results: (1) a fast search method for the optimal subarray positions for a given array specification, and (2) a method for simultaneously optimizing a family of arrays over a range of a given array design parameter. The first search method may be run offline in a few minutes on a desktop computer, or may be run interactively with input from the designer. The results from this method are compared to known global optimal solutions generated previously by exhaustive search on a supercomputer cluster. In these cases, the fast search method almost always found the global optimum in just a few minutes. the second method, for optimizing array configurations over a range of a design parameter, is demonstrated for optimizing a family of arrays over a range of total array size. It generates near-optimal configurations over the range of array sizes. The families of antenna configurations generated using this second method may also be explored interactively with a graphical interface.

European Conference on Antennas and Propagation (EuCAP)

VERY FAST SUBARRAY POSITION CALCULATION FOR MINIMIZING SIDELOBES IN SPARSE LINEAR PHASED ARRAYS

Christopher Lee[1], **Darren Leigh**[1], **Kathy Ryall**[1], **Hiroaki Miyashita**[2], **and Kazufumi Hirata**[2]

[1]*Mitsubishi Electric Research Laboratories, 201 Broadway, Cambridge, MA 02139 USA. {lee,leigh,ryall}@merl.com*
[2]*Antennas Technology Department,Information Technology R&D Center, Mitsubishi Electric Corporation, 5-1-1 Ofuna, Kamakura, Kanagawa 247-8501, Japan {miyas,kazufumi}@isl.melco.co.jp*

ABSTRACT

We have developed a set of algorithms and software tools to help an antenna designer interactively explore the space of possible subarray positions in a sparse linear phased array, for the purpose of designing antennas that minimize the maximum sidelobe level of the array response. This paper presents two new results: (1) a fast search method for optimal subarray positions for a given array specification, and (2) a method for simultaneously optimizing a family of arrays over a range of a given array design parameter. The first search method may be run offline in a few minutes on a desktop computer, or may be run interactively with input from the designer. The results from this method are compared to known global optimal solutions generated previously by exhaustive search on a supercomputer cluster. In these cases, the fast search method almost always found the global optimum in just a few minutes. The second method, for optimizing array configurations over a range of a design parameter, is demonstrated for optimizing a family of arrays over a range of total array size. It generates near-optimal configurations over the range of array sizes. The families of antenna configurations generated using this second method may also be explored interactively with a graphical interface.

Key words: Phased array antenna; Synthesis; Optimisation.

1. INTRODUCTION

We have developed a set of algorithms and software tools to help an antenna designer interactively explore the space of possible subarray positions in a sparse linear phased array (Fig. 1), for the purpose of designing antennas that minimize the maximum sidelobe level of the array response. This could, for example, be applied to a distributed array radar problem [1]. This work is part of a larger effort on phased array antenna design using human guided search [2] to build systems that leverage the complementary strengths of computers and human designers to design high-performance antenna configurations.

This paper presents two new results: a fast search method for optimal subarray positions for a given array specification (Sec. 2), and a method for simultaneously optimizing a family of arrays over a range of a given array design parameter (Sec. 3).

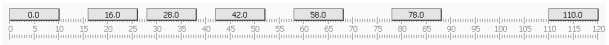

Figure 1. A typical configuration of the type of array we are studying. The two outer subarrays are fixed, to constrain the problem size. The inner subarrays are positioned inside, without overlap, and must be on half-wavelength boundaries. Each element is labeled with its distance, in wavelengths, from the beginning of the array.

2. FAST ARRAY OPTIMIZATION

2.1. Previous work: generate, evaluate, filter, browse

Our previous work on antenna design is based on a generate-then-browse approach, as summarized in Fig. 2. We built a system [3, 4] that discretized the design space, exhaustively generated and evaluated all possible designs in this discretized space, and stored the best 1 million configurations. An antenna designer could then use the browser shown in Fig. 3 to interactively explore this pre-filtered design space. The evaluation routine computes the gain pattern for each configuration and finds the width of the main lobe, and the gain and frequency of the first sidelobe and the highest sidelobe. The million configurations with the lowest maximum sidelobe level (normal-

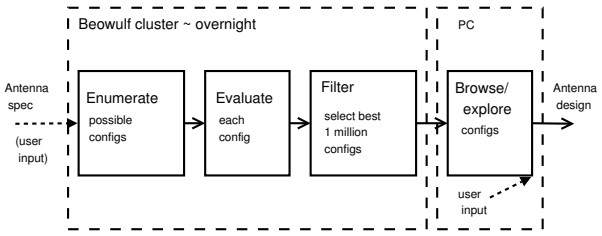

Figure 2. Summary of generate-browse design strategy

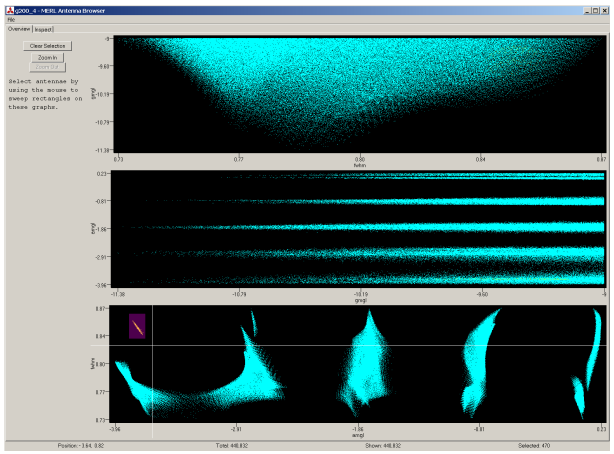

Figure 3. Antenna configuration browser. This browser facilitates exploration of on the order of a million prefiltered array configurations at a time [3].

ized to the main lobe) are stored for display in the interactive browser. Because this method searches the design space exhaustively, the best configurations found are known to be globally optimal.

The arrays we considered have a total size of 120λ, with subarrays 10λ wide composed of 16 isotropic radiators spaced 0.625λ apart (see Fig. 1). They have fixed subarrays at each end, and $4 \leq n \leq 9$ interior subarrays whose positions are discretized to $\lambda/2$ boundaries. The total number of possible configurations for these are shown in Tab. 1 (see Appendix). Because of the large number of possible configurations, exhaustive generation and evaluation requires a high-performance supercomputer cluster. Larger design spaces cannot be easily investigated using exhaustive generation. This motivated the development of much faster ways to generate sets of candidate candidates for browsing.

2.2. Fast search method

To speed the process of finding good candidate configurations of subarray positions, we developed a a simple greedy algorithm to rapidly find large numbers of local minima sampled evenly over the design space. Let $\mathcal{P}$ be the set of possible configurations given the design parameters of the array, and let $c(p)$ be the cost of a given configuration $p \in \mathcal{P}$ (in this case, $c(p)$ is the gain ratio of the maximum side lobe to the main lobe). The method is:

1. Initialize the set of visited configurations to empty $\mathcal{V} \leftarrow \emptyset$.
2. Pick a random configuration: $p \leftarrow \text{rand}(\mathcal{P})$.
3. If it is already visited ($p \in \mathcal{V}$), go to step 2. Otherwise, mark it as visited: $\mathcal{V} \leftarrow (\mathcal{V} \cup p)$.
4. Let $\mathcal{N}$ be the set of configurations that can be reached from p by moving a single subarray a single step (allowing a subarray to "push" another if they are touching).
5. Evaluate each element in $\mathcal{N}$, and pick the best: $n \leftarrow \text{argmax}_{n'}(c(n'))|_{n' \in \mathcal{N}}$.
6. If $c(p) < c(n)$ (a local minimum has been reached) then go to step 2. Otherwise, $p \leftarrow n$ and go to step 3.

	# of interior subarrays				
	4	5	6	7	8
Possible configurations (millions)	9.3	97	470	870	380
Minutes elapsed before best configuration found	< 1	4	*(A)*	4	< 1

Table 1. Approximate number of possible discretized configurations for each number of interior subarrays, and minutes of search on a desktop computer before the best single configuration was found. (A) For 6 inner subarrays, no better than 3rd-best configuration was found in 20 minutes of search. The 3rd best configuration was found within 2 minutes.

The search is an anytime algorithm, and may thus be terminated after a given amount of search time, a given number of configurations have been visited, or when the entire configuration search has been visited. It can be further sped by caching the results of evaluated configurations.

To be effective in covering the search space, the selection of a random state must sample the design space uniformly. This can be done by enumerating the possible discrete configurations, then using this enumeration to map a uniformly sampled integer to a configuration. The enumeration is described in the Appendix.

2.3. Optimization Results

We can tell how well this search method finds optimal candidate configurations, for at least some array parameters, because we can compare it to results previously obtained from exhaustive search on a supercomputer cluster. We ran the search on a desktop computer for arrays of total size 120 wavelengths and between 4 and 8 interior subarrays. As shown in Tab. 1, the global optimal configuration is typically found within a few minutes on a desktop computer. Fig. 4 shows the proportion of the best 10 and best 100 configurations found for the array with 7 inner subarrays (the array with the largest search space tested). We can see that roughly half of the top 10 configurations and a sixth of the top 1000 configurations are found within 20 minutes on a desktop computer.

2.4. Interactive search

The search method of Sec. 2.2 can also be interspersed with human designer input. Fig. 5 shows a graphical in-

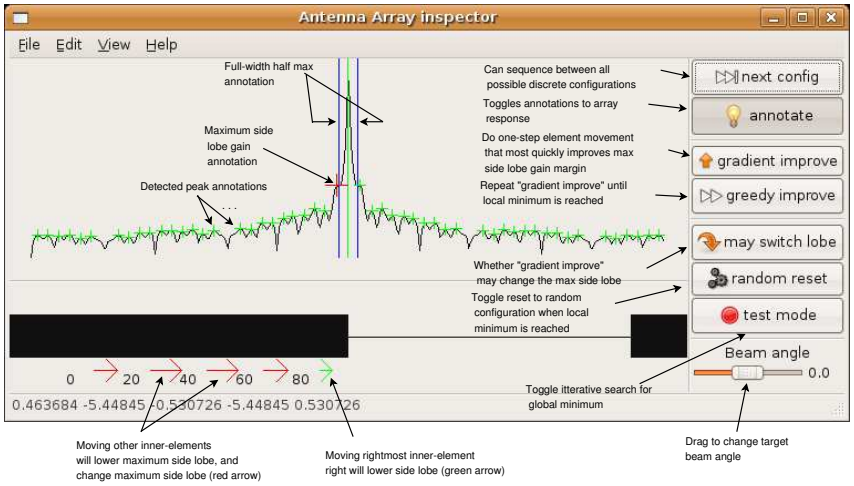

Figure 5. Graphical interface for fast, interactive antenna optimization

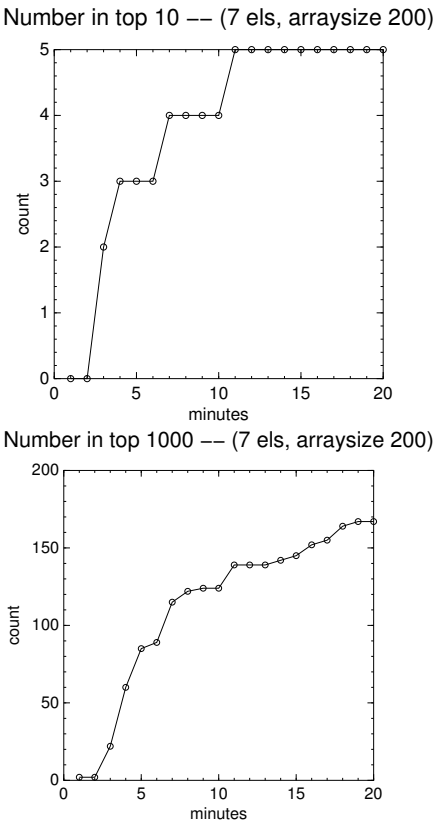

Figure 4. Number of top 10 and top 1000 best configurations found, verses search time. This is for arrays with 7 subarrays, out of 869,648,208 possible configurations.

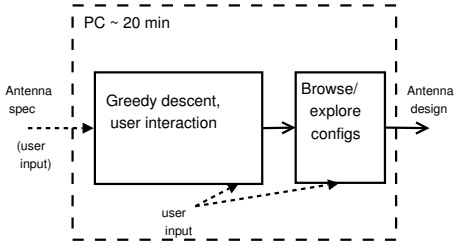

Figure 6. Revised design strategy, with fast, interactive candidate generation

terface with which a designer may switch between automatic improvement and user exploration and manipulation. The designer can directly manipulate subarray positions and see the resulting gain patterns, see a display of which subarrays might be moved to improve performance, or tell the computer to take one or more steps automatically. The resulting configurations can be saved and browsed using the interface shown in Fig. 3.

3. OPTIMIZING FAMILIES OF ARRAYS

Although the fast search method described in Sec. 2 allows a designer to start browsing many optimal and near-optimal results after just a few minutes of computation, this is just for a single set of antenna design parameters (e.g., number of interior subarrays, number of individual radiators each subarray, subarray size, total array size, and fill-factor). This section introduces a method for computing near-optimal configurations over a *range* of a continuous design parameter. For now, we allow a range only for one parameter at a time, and have so far focused on optimizing over a range of total array sizes. These solutions for "families" of arrays may be used to

- quickly explore the design parameter space (e.g., determine the best total array size to use), and thus

solve more open-ended design problems, or

- generate near-optimal configurations extremely quickly (e.g., for use in a microcontroller) over the set of parameters.

A high level description of the method is:

1. For the largest value in the specified range of total array sizes, find a set of "seed" configurations by running the fast search algorithm of Sec. 2, and filtering them so they are sufficiently different from one another (Sec. 3.1).
2. For each seed configuration, generate an initial mapping function from array size to subarray positions, with a set of adjustable coefficients (Sec. 3.2).
3. For each initial mapping from the previous step, adjust the coefficients to optimize the performance of each mapping over the entire range of array sizes (Sec. 3.3).
4. Compare and inspect the optimized models, and select one or more (Sec. 3.4).

3.1. Generating seed configurations

The first part of the process is to generate "seed" configurations. We use configurations optimized for a single value of the range parameter (total array size, via the method of Sec. 2). We typically perform this initial optimization at the highest end of the array size range, because this is the most difficult part of the space to optimize.

The method of Sec. 2 is run (for about twenty minutes) to generate a variety of near-optimal configurations. Then, the resulting configurations are filtered so as not to be too similar to one another, to avoid multiple seeds leading to the same local minimum:

1. An initial population of configurations $\mathcal{P}$ is generated by taking the best few hundred results from a run of fast array optimization at $x = x_{\max}$.
2. The set of seed configurations is set to empty $\mathcal{S} \leftarrow \emptyset$.
3. For each element $\mathbf{p}_i \in \mathcal{P}$ in decreasing order of fitness ($c(\mathbf{p}_i) \leq c(\mathbf{p}_{i+1})$): If $[D(\mathbf{p}_i, \mathbf{s}) > \epsilon]_{\forall \mathbf{s} \in \mathcal{S}}$, then $\mathcal{S} \leftarrow (S \cup \mathbf{p}_i)$.

Here the distance function $D(\mathbf{p}_i, \mathbf{s})$ is the minimum number of one-element steps necessary to transform $\mathbf{p}_i$ into $\mathbf{s}$.

3.2. Parameterizing the mapping

This step takes a single seed configuration and builds a parameterized mapping over the entire range of the variable parameter (i.e., total array size). This is a mapping from a value of the variable parameter to the position of each subarray within the array. This mapping will be further optimized over that range in the next step.

Canonical position Because we want this mapping to represent all physically realizable configurations and no physically unrealizable configurations (i.e., subarrays may not overlap, and no subarray may lie outside the antenna), we chose a simple canonical representation for the locations of the subarray elements. For each subarray in order from left to right, its canonical position is 0 at its leftmost possible location, 1 at its rightmost possible location, and linearly scaled in between. So an antenna with four interior subarrays all bunched at the leftmost side of the array, the canonical position is $[0\ 0\ 0\ 0]$. In practice, if an element of a canonical position vector is not within the range 0 to 1, it is clamped to that range before use. This is a continuous parameterization.

Polynomial mapping The mapping from array size to subarray positions is represented by a set of polynomials, one for each subarray. Polynomials were chosen for the mapping functions due to their smoothness and simplicity. If p_i is the canonical position of the ith element from the left of the array, where x is the parameter value and x_s is the value of that parameter used in generating the seed value, then the position mapping is:

$$p_i = a_{i0} + a_{i1}(x - x_s) + a_{i2}(x - x_s)^2 + \ldots + a_{im}(x - x_s)^m \quad (1)$$

or

$$\mathbf{p} = \mathbf{f}(\mathbf{a}, x). \quad (2)$$

The polynomial coefficients $\mathbf{a} = \{a_{ij}\}$ will be trained to optimize the quality of the output positions over the range of the variable parameter (i.e., array size). Because the initial polynomial mapping is taken from a seed position $\mathbf{s}$, we set $a_{i0} \leftarrow s_i$, and $a_{i,\{j \neq 0\}} \leftarrow 0$. This causes the array to be at the seed position when $x = x_s$ (when the array is the size that the seed position was optimized for). Because $\mathbf{p}$ represents a canonical position, the positions of the subarray elements are initially scaled proportionally to the total size of the array in this mapping function.

3.3. Optimizing the mapping

Once the initial mapping is created, we can optimize it to improve the performance of the array configurations it generates. The cost function chosen for the optimization considers the individual costs of the generated configurations over the range of the variable parameter $x_{\min} \leq x \leq x_{\max}$.

$$C(\mathbf{a}) = \int_{x_{\min}}^{x_{\max}} \frac{c(\mathbf{f}(\mathbf{a}, x))}{E[c(x)]} dx \quad (3)$$

The normalization value, the expected performance of an array with the given parameter value ($E[c(x)]$), is estimated by averaging the performance of randomly selected array configurations with that parameter value. It does not depend on $\mathbf{a}$. In practice, the above integral is approximated by sampling evenly over the range of the variable parameter.

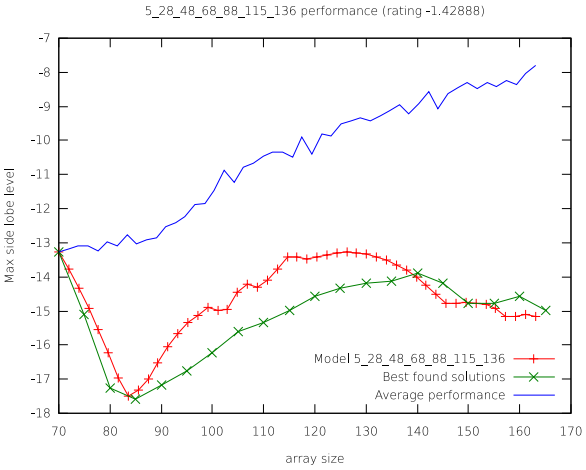

Figure 7. Performance of mapping for arrays with 7 inner elements. The maximum sidelobe level is given in dB, and the array size in wavelengths (λ, not *including the lengths of the exterior subarrays, which are each 10λ). The mapping is labeled by the index numbers of the subarray positions of its seed configuration.*

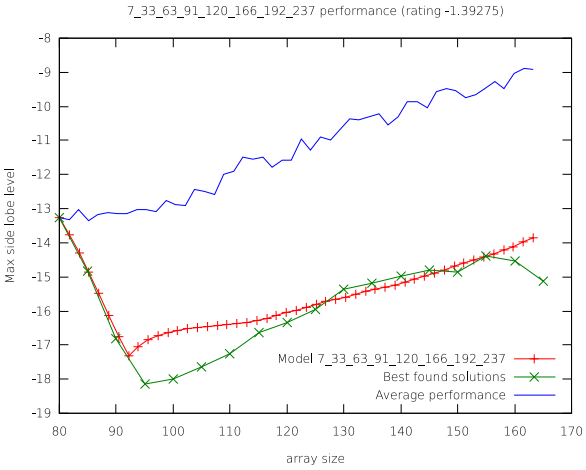

Figure 8. Performance of mapping for arrays with 8 inner elements.

It is not easy to compute the gradient of (3), so we chose an optimization method that does not require a gradient: the simplex algorithm of Nelder and Mead [5]. Because the output of mapping function $f(c, \cdot)$ is much more sensitive to the coefficients of higher-order polynomial terms than to lower-order terms, we scale the coefficients by a factor ρ so they will have roughly the same effect before they are passed to the optimizer. Here, s is the distance between the exterior subarrays, n is the number of subarray elements, and d is the degree of a coefficient's term:

$$\rho(s, n, d) = \frac{(s/n)^{d+1}}{2.0} \tag{4}$$

3.4. Evaluating/browsing the mapping

Because each seed configuration results in a different output mapping, we run the optimization for a set of about 10 distinct seeds, and select from the resulting mappings. These can be compared by the value of the cost function they achieved. If there are a few output mappings with similar resulting cost values, they can also be compared via plots for how they perform over the range of parameter values, and by browsing the array configurations they generate over the range of parameter values.

We ran this optimization algorithm for arrays with between 4 and 9 interior subarrays Each subarray is 10λ wide and composed of 16 isotropic radiators spaced 0.625λ apart and approximately uniformly illuminated. The total array size was set to the range of 80λ (or the minimum array size that fits the required number of subarrays) to 190λ. We generated mapping functions using cubic polynomials.

Our algorithm typically yielded results less than one decibel RMS from sampled estimates of global optima. Fig. 7 and Fig. 8 show performance of two of the best generated mappings for arrays with 7 and 8 interior subarrays respectively. The model line is the sampled performance over the range of array sizes. The "best-found solutions" line in the plot is an attempt to compare the performance to best possible at each parameter value. It is generated by running the quick search algorithm of Sec. 2 for 40 minutes at each sampled array size. For smaller arrays this is probably the global optimum or close to it, but for the larger arrays, where the number of possible configurations is far higher, this is probably not the global optimum. In fact we see that in some cases the mapping results in better configurations than the best found after 40 minutes of the fast discrete space search.

The topmost lines in the plots are estimates of the expected performance of randomly selected arrays of the given total size. This is the normalization value $E[c(x)]$ used in the cost function (3).

One interesting result is that both plots give an indication of what the best overall size might be for these array families. The arrays with 7 inner elements have the lowest maximum side lobe between 80λ and 90λ (between total lengths of 100λ and 110λ including the lengths of the exterior subarrays). The arrays with 8 inner elements have the lowest maximum side lobe between 90λ and 100λ (between total lengths of 110λ and 120λ including the lengths of the exterior subarrays). This kind of information only becomes apparent when you can optimize over a range of design parameters.

The resulting models from the optimization can also be browsed using the graphical interface shown in Fig. 9. This browser lets the user both directly manipulate subarray positions and change the variable parameter and see the resulting array configuration and corresponding gain patterns.

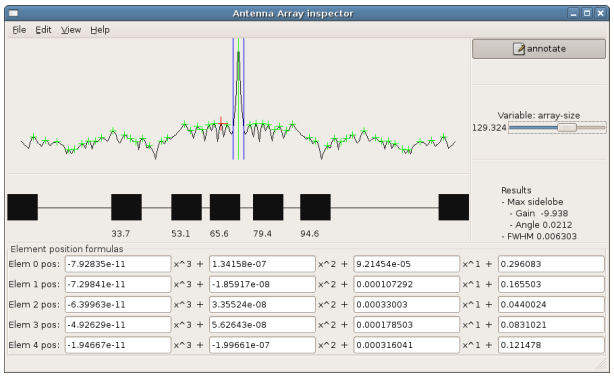

Figure 9. Graphical interface for browsing array configurations generated by optimized mappings

4. CONCLUSION

This paper presented two new results for helping designers of sparse linear phased array antennas find subarray configurations that minimize side lobes of the gain pattern. The first result is a fast search method for optimal subarray positions for a given array specification. It eliminates the previous need to use a supercomputer cluster to generate and filter the configuration space the designer will browse. For the kinds of arrays we investigated, it typically finds the global optimum configuration in a few minutes. The second result is a method for simultaneously optimizing a family of arrays over a range of a total array sizes. It typically yields results less than one decibel RMS from sampled estimates of global optima over the range of antenna sizes, and allows a designer to quickly find answers to questions about design parameters such as how large an array should be.

REFERENCES

1. R. C. Heimiller, J. E. Belyea, and P. G. Tomlinson. Distributed array radar. *IEEE Transactions on Aerospace and Electronic Systems*, AES-19(6), November 1983.

2. MERL project: Human-guided antenna design. `http://www.merl.com/projects/antenna/`.

3. D. Leigh, K. Ryall, T. Lanning, N. Lesh, H. Miyashita, K. Hirata, Y. Hara, and T. Sakura. Sidelobe minimization of uniformly-excited sparse linear arrays using exhaustive search and visual browsing. In *IEEE Antennas and Propagation Society International Symposium (AP-S International Symposium)*, volume 1B, pages 763–766, 2005.

4. D. Leigh, T. Lanning, N. Lesh, K. Ryall, H. Miyashita, and S. Makino. Exhaustive generation and visual browsing for radiation patterns of linear array antennas. In *ISAP*, Sendai, Japan, August 2004.

5. J.A. Nelder and R. Mead. A simplex method for function minimization. *Computer Journal*, 7:308–315, (1965.

APPENDIX: ENUMERATING DISCRETIZED ARRAY CONFIGURATIONS

Enumerating the possible discretized positions of subarrays in a sparse phased array antenna is useful for computing the size of the design space, and for generating random configurations. One such enumeration is:

1. Let configuration 0 be that in which all interior subarrays are bunched at the left side of the array.
2. While possible, move the rightmost subarray one step right and assign the new configuration the next ordinal value.
3. When the rightmost subarray reaches the right side of the array, move one step to the right the rightmost subarray that *can* be moved, bunching against it all subarrays to its right. Assign this new configuration the next ordinal value, and go to step 2. If no subarrays can be moved to the right, then all configurations have been enumerated.

The following algorithm can map a given index to a set of (non-canonical) positions quickly.

Compute discrete subarray position $\mathbf{p}$*, given position index* I*, distance between the exterior subarrays* s*, number of subarray elements* n*, and width of a subarray* w *(all integers)*:

$l \leftarrow 0$ *(leftmost available position)*
for i in $0 \ldots n-1$ *(loop over elements left to right)*
  $r \leftarrow s - w(n-i)$ *(available space right of element)*
  $p_i \leftarrow l$
  for j in $0 \ldots r-1$
    $d \leftarrow z(w, n-i-1, s-l-w-j)$
    if $d \leq I$
      then $p_i \leftarrow l+j+1$; $I \leftarrow I-d$
      else $l \leftarrow p_i + w$; next i.

$z(w,n,s)$ is the number of possible configurations for an array with n inner elements of width w and array size (space between the exterior subarrays) s.

$$z(w,n,s) = \begin{cases} 0 & \text{if } s < nw, \\ s-w+1 & \text{if } n = 1, \\ \sum_{i=0}^{s-w} z(w, s-w-i, n-1) & \text{otherwise.} \end{cases} \quad (5)$$

This recursive formula can be computed efficiently using dynamic programming.