Learning to Predict Noise Level for PWB Layout

Paul E. Utgoff
Peter Stephen Heitman

COINS Technical Report 87-118

November 5, 1987

Department of Computer and Information Science
University of Massachusetts
Amherst, Massachusetts 01003

Telephone: 413-545-4843

Abstract

This paper describes an algorithm for learning to predict noise levels for candidate transmission line networks in printed-wire-board layout. The algorithm, implemented in the program SURVEYOR, incrementally refines n-dimensional interpolating surfaces for specified locations of interest in a network. The program includes an experiment management strategy that automatically picks new points for which it is advantageous to obtain the noise level via circuit simulation. This is important for refining each surface near the maximum acceptable noise level, and not refining each surface where accuracy is not needed. Empirical results are presented that illustrate building of a noise level surface and refinement near the noise margin.

Contents

1. The Problem **1**

2. A Solution **1**

2.1 Initializing the Surface 2

2.2 Refining the Surface 4

2.3 Picking Points for Refining Near the Noise Threshhold 7

3. An Experiment: Refining a Contour in 2 Dimensions **8**

4. Summary **9**

1. The Problem

The problem addressed here is that of building a computer program that quickly and accurately predicts whether the maximum noise level for any given transmission line of a candidate network exceeds its fixed acceptable noise threshhold. Here, *network* refers to a connected set of transmission lines, terminated at devices. The devices, lengths of transmission lines, and other parameters of the network impact the noise level at various points in a network. The issue here is to determine which combinations of parameters for a given network produce specific noise levels at points of interest in the network(Ginzburg and Shen, 1986). Throughout the discussion, the term *specific network* refers to a network with a specific set of parameters. The principal methods for predicting noise levels either simulate a specific network or interpolate from previously performed simulations. By constructing a contour map, hereafter called a *noise surface*, it is possible to interpolate noise level for specific networks that have not been simulated. Until now, the noise surface has been constructed manually. There are two important subproblems. First, there is need of an algorithm that refines a noise surface incrementally. Second, given that network simulations are necessary for determining noise levels and that such simulations are expensive[1] there is need of an orderly automatic strategy for deciding which specific networks should be simulated. The SURVEYOR program was built to address these needs.

The rest of the paper addresses the generic problem of how to automatically construct an individual noise surface for a specified point of interest in a network. The problem is stated as follows:

Given:

- The ability to represent each specific network entirely as a set of numeric parameters.
- A circuit simulator that returns the maximum noise level at specified points of interest[2]
- A fixed noise threshhold below which the network is acceptable.

Find:

- A fast predictor of noise level that is accurate with respect to the noise threshhold.

2. A Solution

The approach taken here is to model the noise level by an n-dimensional interpolating function. A specific requirement is that the model be updated incrementally whenever the noise level is determined for the network parameters. Being given the noise level (function value) at for a set of network parameters (point) should require refinement of the interpolation surface, not generation of a new surface. A second important reason for

[1]requiring from half to several minutes each

[2]The noise level for the other points of interest would also be used to refine the corresponding noise surfaces.

requiring an incremental algorithm is that the decision strategy for choosing which point (specific network) to simulate depends upon the way in which the surface develops. Each new point reveals some part of the noise surface. Given that accuracy is needed only near each noise threshhold contour but not far from each one, it is not possible to decide *a priori* which points should be simulated. It was considered unacceptable to simply request simulations at each grid point of a sufficiently fine grid. One wants to use the expensive simulator sparingly, only where necessary to achieve the desired accuracy.

A new interpolation algorithm was devised. There are two reasons for inventing yet another interpolation algorithm. First, no existing algorithm was uncovered that permits incremental refinement of the interpolating function in a localized region. For example, a version of Shepard's Method was ruled out because each new point has a global effect on the entire surface(Schumaker, 1976). Furthermore, uneven density of known points induces inaccuracy in areas with fewer known points. Second, polynomial functions were ruled out due to the inaccuracy of the high degree expressions. It was decided to try linear interpolation in n dimensions. Such a surface can be refined incrementally in a localized area of interest, leaving the rest of the surface unaffected.

The interpolation algorithm devised for SURVEYOR is called the Simplex Interpolation Algorithm. The rest of this section is organized as three parts. The first part discusses the initialization of the surface. The second discusses incremental refinement of the surface. The last part describes the strategy for deciding which points should be simulated so that the surface is refined where necessary.

2.1 Initializing the Surface

The domain of an n-variable function, with each variable restricted to a value within a closed interval, defines a hypercube, where ***hypercube*** is a cube in any dimensionality. The extent of the hypercube in a given dimension is limited by the minimum and maximum possible values of the variable that defines that dimension. The surface is always defined as a set of simplexes (and function value at each vertex) whose union is equal to the hypercube. The initialization problem is to generate the initial set of simplexes, also called a ***simplicial-complex.***

A ***simplex*** is the set of points bounded by a simplest polyhedron in the dimensionality (Aleksandrov et al, 1956). For example, a 1-dimensional simplex is a line segment, a 2-dimensional simplex is a triangle, and a 3-dimensional simplex is a tetrahedron. A simplex in $n + 1$ dimensions is the set of points bounded by a simplest polyhedron that contains a simplex in n-dimensions and a point not contained in the n-dimensional simplex. It is this contructive method that forms the basis of the initialization algorithm.

The initialization algorithm generates the initial simplicial-complex in n dimensions by extending the initial simplicial-complex from $n - 1$ dimensions. In the case of $n = 1$ dimension, the simplicial-complex is $\{((0)(2))\}$. The syntax for describing a simplicial-complex is $\{simplexes\}$, with each simplex given as $(vertices)$, with each vertex given as $(index_1, ..., index_n)$. In the initialization, each index has the value 0, 1, or 2, indicating minimum, mid-value, and maximum of a variable respectively. After the initial simplicial-complex is constructed, the values 0, 1, and 2 are replaced by the actual minimum, mid-value, and maximum of the corresponding variable. The correspondence is defined only after the initial simplicial-complex has been built. The final step in the initialization is to simulate each distinct vertex, the variable values defining a specific network.

The initialization algorithm is:

1. Set n to 1. Set the initial simplicial-complex SC to $\{((0)(2))\}$.

2. If n is equal to the number of variables (number of dimensions), goto 6.

3. For each simplex (a face in $n+1$ space), compute the other faces in the $n+1$-dimensional hypercube by performing all parallel insertions of 0 and 2. A parallel insertion is the inserting of a value at the same insertion point in each and every vertex. Define *FACES* to be the set of such faces.

4. Construct the midpoint of the $n+1$ hypercube as $(1,1,...,1)$. Set SC to contain the set of simplexes obtained by adding the midpoint to each face in *FACES*.

5. Increment n. Goto 2.

6. Pick an order of the n variables. For every vertex of every simplex, replace the 0, 1, and 2 indexes, from left to right, with the minimum, mid-value, and maximum of the corresponding variable.

7. Simulate each specific network that corresponds to a distinct vertex in any simplex in the simplicial-complex.

An example illustrates the algorithm. Assume two variables, $x \in [10,20]$ and $y \in [30,40]$. Step 1 of the initialization algorithm defines SC to be:

$$\{((0)(2))\}$$

Step 3 defines *FACES* to be

$$\begin{aligned}\{ \quad & ((0,0)(0,2)),((0,0)(2,0)),\\ & ((2,0)(2,2)),((0,2)(2,2)) \quad \}\end{aligned}$$

This is done by inserting 0 at the left of each vertex in $((0)(2))$, and by inserting 0 at the right of each vertex, 2 at the left of each vertex, and 2 at the right of each vertex. Geometrically, the initial simplex, a line segment, is converted to the four faces of the rectangle. Step 4 defines the midpoint (11). By adding the midpoint to each face in *FACES*, SC becomes

$$\begin{aligned}\{ \quad & ((0,0)(0,2)(1,1)),((0,0)(2,0)(1,1)),\\ & ((2,0)(2,2)(1,1)),((0,2)(2,2)(1,1)) \quad \}\end{aligned}$$

Geometrically, the midpoint of the rectangle is added as a new vertex to each face, converting each face to a triangle. Next, step 6 substitutes the corresponding variable values, producing the simplicial-complex

$$\begin{aligned}\{ \; &((10,30)(10,40)(15,35)),((10,30)(20,30)(15,35)),\\ &((20,30)(20,40)(15,35)),((10,40)(20,40)(15,35)) \; \}\end{aligned}$$

Finally, the specific network corresponding to each distinct vertex is simulated. The distinct verteces are $(10,30)$, $(10,40)$, $(15,35)$, $(20,30)$, and $(20,40)$.

Before discussing refinement of the surface and the strategy for picking points at which to simulate the layout, consider how the surface is used to interpolate at a point.

To interpolate at a point, find a smallest simplex that contains the point. Next, construct a linear interpolating surface from the vertices of the simplex. Because each simplex in an n-dimensional surface has $n+1$ vertices, and each vertex with its function value represents a point in $n+1$-dimensional space, it is possible to construct an n dimensional planar surface passing through those $n+1$ points. It is then a simple matter to determine the function value of the given point by determining the intersection of a line passing through the point (parallel to the function value axis) and the planar surface. Notice that the surface defined by simplexes (and function value at each vertex) provides a continuous interpolating surface. If the given point lies on the boundary of two simplexes, it does not matter which of the simplexes is used for the interpolation. This is because using linear interpolation, the function value at any point on the boundary is determined by the function values of the vertices defining the boundary.

2.2 Refining the Surface

The surface can be refined by obtaining the function value at a given point, and decomposing the smallest simplex containing the point into a set of smaller simplexes, using the given point as a new vertex. This works[3] for any face of the containing simplex that does not also contain the new point, because combining a face which spans $n-1$ dimensions with a distinct point again forms an n-dimensional simplex.

The algorithm for refining the surface at a point is:

1. First, the simplexes containing the point are identified. Note that there may be more than one if the point is contained in a face of a simplex.

2. Construct new simplexes by combining the new point with each face of the containing simplex, excluding any face that contains the new point.

3. Record each new simplex as a refinement of the containing simplex. This is for the sake of efficiency. By storing simplexes in a containment hierarchy, the computational complexity of locating a smallest containing simplex is reduced significantly.

[3]when the new point is truly new and is not at an existing vertex. This is always the case for the Simplex Interpolation Algorithm.

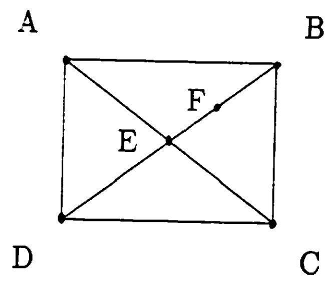

Figure 1: Initial simplexes for $2D$ Hypercube

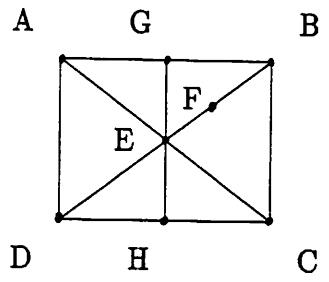

Figure 2: Hypercube Circumscribing Simplex

Consider an example for a 2-dimensional surface. As shown in figure 1, the simplexes of the initial surface consist of 4 triangles. The initial set of simplexes is a linear list because no refinement has yet occurred. Observe the refinement process when the value at point F on $\overline{BE}$ is obtained.

First, the refinement algorithm identifies those simplexes that contain the point F. Each simplex in the initial surface needs to be tested to determine if it contains F. For efficiency reasons, the test for containment has two parts. The first test determines whether the point lies within the hypercube that circumscribes the simplex. As shown in figure 2, $\triangle ADE$ cannot contain point F since it is not even contained in $\Box AGHD$. If the point lies within the containing hypercube, a second, more accurate, test is employed. This test uses vector cross products to determine whether the point lies within the simplex. Cross products are used to determine, for each face of the simplex, whether the point is on the same side of the face that the other points of the simplex are on. The point's position relative to each face must be checked before it is known whether the simplex contains the point. The algorithm used by the second test is shown in figure 3.

Second, if the point is contained in a simplex, check to see if there are any simplexes embedded hierarchically within this one. If there are, then it must be further determined which of the embedded simplexes contain the point. Continue the search down the containment hierarchy for the set of atomic (not decomposed) simplexes that contain the point. For this example, the set of simplexes is $\triangle ABE, \triangle BEC$.

Finally, refine each such simplex using F as the new vertex. Specifically, having deter-

```
Given a simplex $S$ and a point $P$:

  1. for each face $T$ of $S$:
       (a) select a vertex $V$ of $T$ to be the anchor-point
       (b) form the set of vectors $\mathbf{W}$ by subtracting $V$ from all of the other vertices of $T$
       (c) form the cross product $\vec{CP}$ of all of the vectors in $\mathbf{W}$
       (d) form the vector $\vec{VP}$ by subtracting $V$ from $P$
       (e) if $\vec{VP} \cdot \vec{CP} = 0$
           then $P$ lies on $T$, go to 2
           else
              i. select a vertex $N$ of the simplex that is not on $T$
             ii. form the vector $\vec{VN}$ by subtracting $V$ from $N$
            iii. if the sign of $\vec{VN} \cdot \vec{CP}$ is different from the sign of $\vec{VP} \cdot \vec{CP}$
                 then $P$ does not lie in $S$, go to 3 else $P$ may lie in $S$, continue
       (f) if more faces exist, go to 1
       (g) if all of the faces have been tested, go to 2
  2. $P$ lies in $S$, return true
  3. $P$ does not lie in $S$, return false
```

Figure 3: Algorithm Used by Cross Product Test

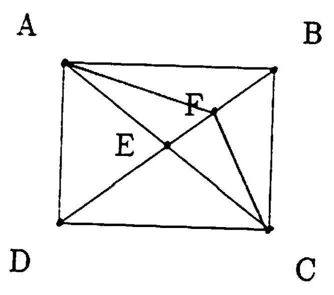

Figure 4: Surface After Adding Point F

mined that point F is contained by $\triangle ABE$, combine F with $\overline{AB}$ to produce $\triangle ABF$, and combine F with $\overline{AE}$ to produce $\triangle AEF$. Note that $\triangle BEF$ is not constructed because it would be empty. Because F is contained in $\overline{BE}$, F is also contained in $\triangle BCE$, so two additional simplexes $\triangle BFC$ and $\triangle ECF$ are also constructed. Figure 4 displays the surface after the refinement from adding point F.

Each new simplex is recorded in the hierarchy as being contained within the simplex from which it was constructed. The containing simplex is not discarded. Due to this hierarchy, if a simplex contains a given point, then one of the contained simplexes must also contain the point. If a simplex does not contain a given point, then none of its contained simplexes can contain the point. This is very efficient when searching for the simplexes containing a point. Rather than having to test every simplex for containment of the point, only those simplexes of the initial surface and the simplexes stored hierarchically within the simplexes containing the point need to be tested. For large numbers of simplexes, particularly as the dimensionality of the surface increases, this leads to a dramatic improvement in performance. The improvement seen will depend on the dimensionality of the surface since in n dimensions each new point divides a simplex into $n+1$ simplexes (or n simplexes if the point lies on a side of the simplex). In one particular test involving 4 dimensions, it was 20 times faster to find the smallest containing simplex using the hierarchical storage of simplexes rather than a linear storage of simplexes.

2.3 Picking Points for Refining Near the Noise Threshhold

Where the surface is well above or well below the noise threshhold, the surface does not need to be refined. At levels near the threshhold, accuracy is needed so that the surface can be used with high reliability to determine whether a layout will be within an acceptable noise margin.

Consider the threshhold as dividing the area into disjoint regions. Each region contains only points that have function values either below or above the noise threshhold, but not both. Each region is bounded by the contour defined by the threshhold. The preliminary phase of the algorithm is to use the circuit simulator to evaluate the noise level at each point of an evenly-spaced grid layed across the surface. The goal of this step is to place at least one point within each region. If the grid is too sparse and regions exist that do not contain a point of the grid, the contours surrounding those regions may not be refined in the subsequent processing. Conversely, if the grid is denser than required to identify a point in each region, the work of determining the function value at the extra points was unnecessary.

After the initial grid is laid, the primary phase is to make several passes through the list of simplexes, refining those simplexes whose values straddle the threshhold, that is, subdividing those simplexes that contain vertices whose function value is below the threshhold and vertices whose function value is above the threshhold. The point used to subdivide the simplex should be such that, over time, the length of the longest line segment that straddles the threshhold could become arbitrarily small. If this were not the case we could not guarantee that the desired accuracy of interpolation could be reached. The point chosen by the Simplex Interpolation Algorithm at which to subdivide the simplex is the midpoint of the longest edge.

During each pass, only those simplexes whose size is above a user-specified threshold are considered for subdividing. Before each new pass, the threshold is set to be 75% of the threshold used for the previous pass. Subdividing only those simplexes above a given size results in an orderly growth of simplexes. The passes through the list of simplexes is terminated when all of the simplexes are 'small enough', that is, when the resolution desired for interpolating function values can be achieved. This is implemented by comparing the current value of the size threshold against a user-specified value. After the size threshold is less than or equal to the user's desired value, then the refinement process halts.

3. An Experiment: Refining a Contour in 2 Dimensions

Figure 5 shows a contour map of a sample surface in 2 dimensions. For this experiment, the circuit is imagined[4] to have 2 variable parameters, hence the surface in 2 dimensions. Each parameter is measured in inches and is allowed to range between 0.0" and 10.0". The noise level for this network ranges from 0.0 to 1.0. The noise theshhold used for this experiment was arbitrarily chosen to be 0.5. Note that the threshhold chosen divides the area into 4 regions, two of which are below the threshhold, and two of which are above the threshhold.

The goal is to build a fast and accurate predictor of whether the noise level at a point of interest in the network would be below the threshhold. One method of achieving the desired accuracy of prediction would be to measure the noise level everywhere in the domain of the network's parameters. The contour map shown was generated by laying a fine grid of points everywhere across the domain and calling a circuit simulator to determine the noise level at each of the 1089 points. The filled-in circles in the figure represent the points at which the circuit simulator was called. For illustration, the contour lines were computed by interpolating from nearby points.

Using fewer calls to the circuit simulator, the Simplex Interpolation Algorithm can produce as accurate a predictor as that built by laying down a fine grid. Figures 6 and

[4]The experiment described here was performed using an implementation of SURVEYOR written in Common Lisp on a Texas Instruments Explorer workstation. The circuit and the circuit simulator were simulated for this experiment.

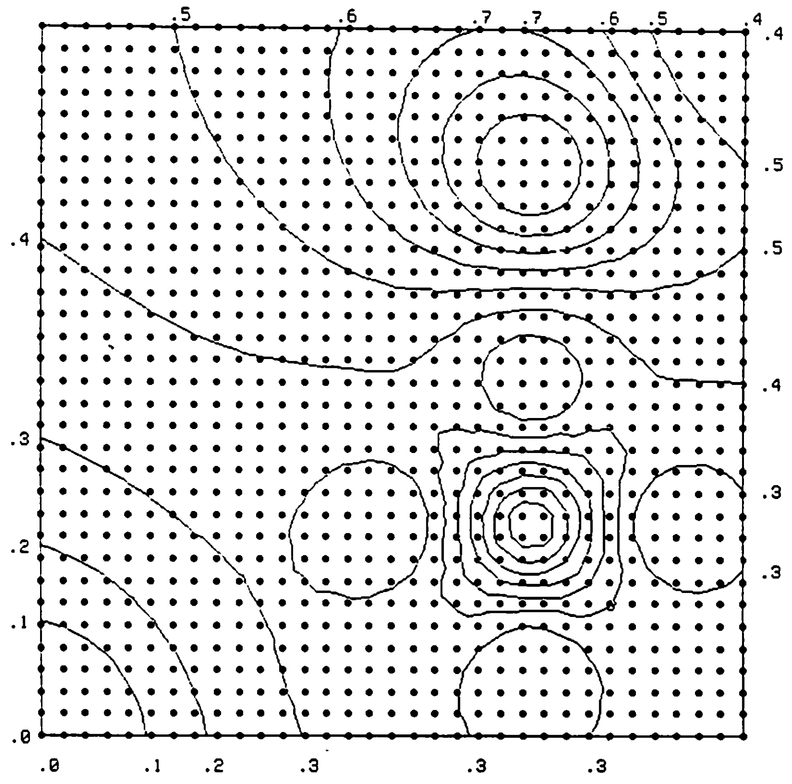

Figure 5: Sample 'Contour Map' of Function in 2 Dimensions

7 show the surface after laying the initial, sparse, grid. Figure 6 displays the contour line corresponding to the threshhold that is found by interpolating on the current surface. Figure 7 displays the simplexes created during the process of laying the initial grid. Note that the function value was needed at 25 points for this first step. The next step is to refine the simplexes that contain vertices both below and above the threshhold. This process is continued until the longest edge of each simplex is less than a specified value, in this case a value of 0.25" was arbitrarily chosen. The final set of simplexes is shown in Figure 9 and the corresponding interpolated threshhold is shown in Figure 8. The noise level at only 200 points were needed to achieve the desired resolution. Observe that calls to the circuit simulator are denser near the threshhold where they contribute to achieving the desired accuracy, but are relatively sparse farther away. This ability to choose points that are useful for increasing the accuracy of the predictor enables the algorithm to make $\frac{1089-200}{1089} = 82\%$ fewer calls to the circuit simulator than were needed for the fine grid. Problems with more dimensions or requiring greater accuracy will experience even greater savings.

4. Summary

SURVEYOR is a computer program that automatically builds a predictor of noise-level at points of interest in networks of transmission lines. Each noise level surface it builds permits generalization to specific networks that have not been simulated. The system learns by refining its noise level surfaces whenever the noise level at a point of

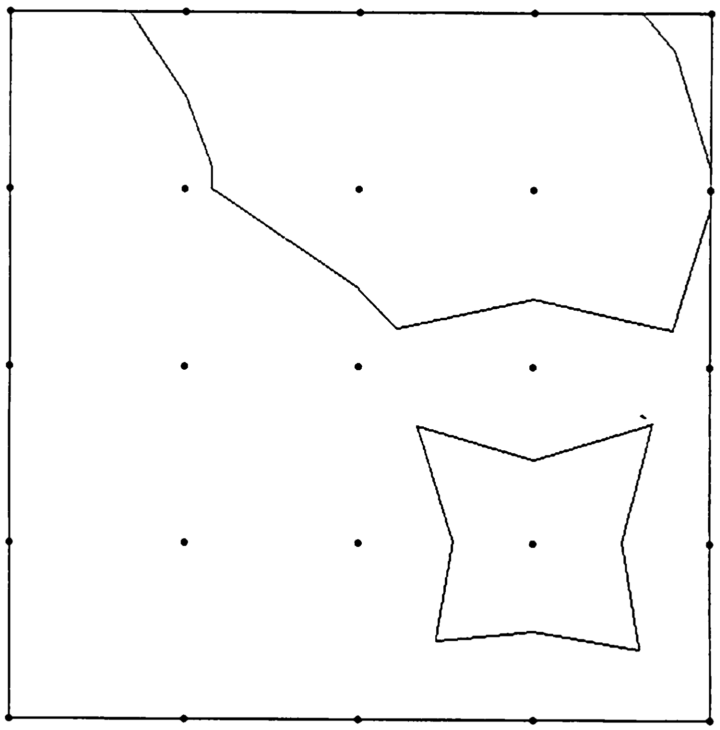

Figure 6: Current Knowledge of Threshhold Contour After Initial Phase in 2 Dimensions

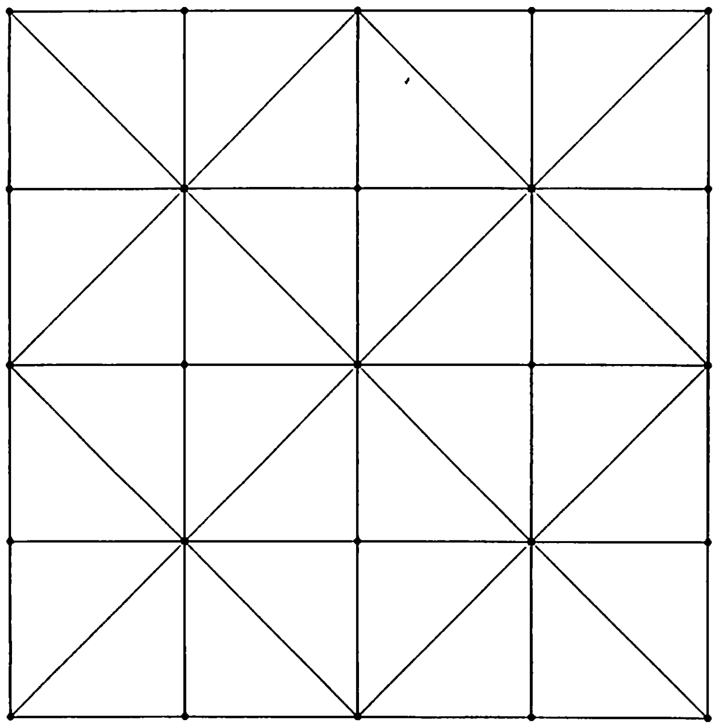

Figure 7: Current Simplex Model After Initial Phase in 2 Dimensions

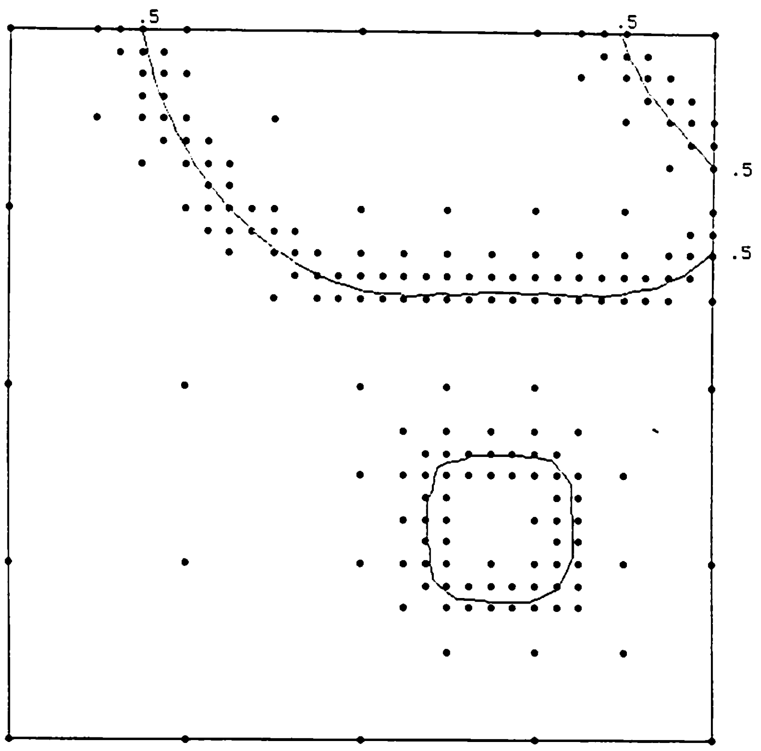

Figure 8: Knowledge of Threshhold Contour After Refining Phase in 2 Dimensions

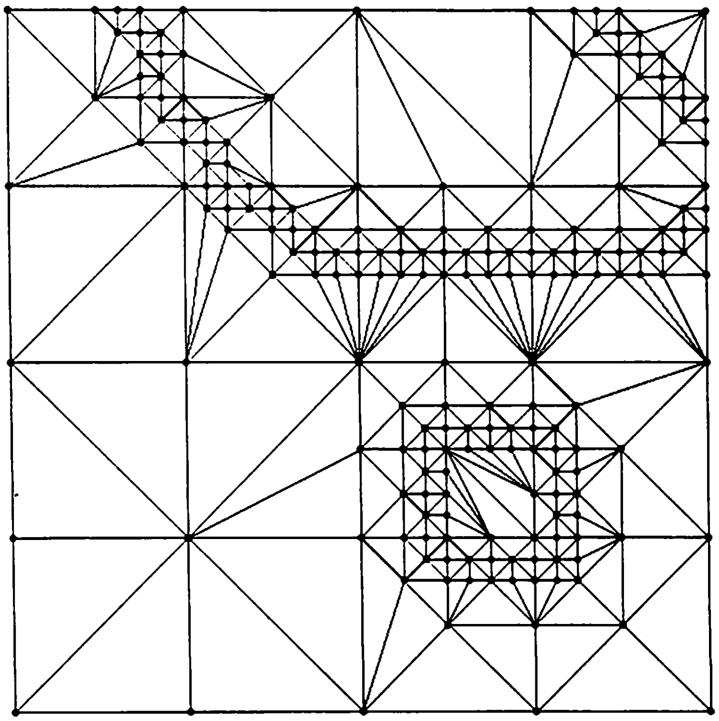

Figure 9: Simplex Model After Refining Phase in 2 Dimensions

interest in a specific network becomes known. SURVEYOR includes a decision strategy for determining which specific networks should be simulated so that a noise surface is refined only where increased accuracy is needed. The net result is an entirely automatic method for constructing noise-level predictors for networks. Such predictors play a critical role within the larger decision making process of PWB layout.

Acknowledgements

This work was supported by a grant from the Digital Equipment Corporation and by the Office of Naval Research through a University Research Initiative Program, under contract number N00014-86-K-0764. Discussions with Olga Ginzburg, Bob Shen, Clay Martin, and Jamie Callan helped immeasurably. Helpful comments and criticisms were provided by Sharad Saxena and Margie Connell.

References

Alexsandrov, A. D., Kolmogorov, A. N. and Lavrent'ev, M. A. (eds) *Mathematics: Its Content, Methods, and Meaning*, vol. III, 1956, MIT Press translation 1977.

Ginzburg, O. and Shen, R. P. "Circuit Simulation and Layout Rules for PCB," unpublished manuscript, May 1986.

Schumaker, L. L. 1976. "Fitting Surfaces to Scattered Data," *Approximation Theory II*, Academic Press, pp. 203-247.