Microcomputers in Civil Engineering **12** (1997) 233–250

Finite-Element Mesh Generation Using Self-Organizing Neural Networks

Larry Manevitz*, Malik Yousef

Department of Mathematics and Computer Science, University of Haifa, Haifa, Israel

&

Dan Givoli

Faculty of Aerospace Engineering, Technion-Israel Institute of Technology, Haifa, Israel

Abstract: *Neural networks are applied to the problem of mesh placement for the finite-element method. When the finite-element method is used to numerically solve a partial differential equation with boundary conditions over a domain, the domain must be divided into "elements." The precise placement of the nodes of the elements has a major affect on the accuracy of the numeric method. In this paper the self-organizing algorithm of Kohonen is adapted to solve the problem of automatically assigning (in a near-optimal way) coordinates from a two-dimensional domain to a given topologic grid (or mesh) of nodes in order to apply the finite-element method effectively when solving a partial differential equation with boundary conditions over that domain.*

One novelty of the method is the interweaving of versions of the Kohonen algorithm in different dimensions simultaneously in order to handle the boundary of the domain properly.

Our method allows for the use of arbitrary types of two-dimensional elements (in particular, quadrilaterals or mixed shapes as opposed to just triangles) and for varying desired densities over the domain. (Thus more elements can be placed automatically near "areas of interest.")

The methods and experiments developed here are for two-dimensional domains but seem naturally extendible to higher-dimensional problems. The method uses a mixture of both one- and two-dimensional versions of the Kohonen algorithm, with an improvement suggested by Tabakman and Exman, and further adapted to the particular problem here. Experimental results comparing this algorithm with a well-known two-dimensional grid-generating system (PLTMG) are presented.

1 BACKGROUND OF THE APPLICATION

The finite-element method (FEM) is a computationally intensive method for solving partial differential equations. Effective use of the method requires setting up the computational framework in an appropriate manner, which typically requires expertise.

In more detail, when applying the FEM to a given domain, one has to divide the domain into a finite number of nonoverlapping subdomains (elements). (In two dimensions, the elements are usually triangles or quadrilaterals.) One also has to define a finite number of nodes, which are the vertices of the elements, and possibly other points as well. The collections of elements and nodes (and the connections among them) constitutes the finite-element mesh, whose quality is an essential ingredient in achieving accurate and reliable numeric results for all finite-element codes. The computational cost of generating the mesh may be much lower, comparable, or in some cases higher than the cost associated with the numeric solver of the partial differential equations, depending

* To whom correspondence should be addressed.

© 1997 *Microcomputers in Civil Engineering*. Published by Blackwell Publishers, 350 Main Street, Malden, MA 02148, USA, and 108 Cowley Road, Oxford OX4 1JF, UK.

on the application and the specific numeric scheme at hand. See, for example, ref. 2 for details.

To achieve a high-quality mesh, one has to (1) decide on the appropriate size and topology of the mesh, (2) decide how it should be placed on the domain, and (3) afterwards make decisions regarding the organization of the data on the mesh that have an affect on the ease of computation. Each of these areas requires expertise.

In this paper we apply the methods of neural networks, in particular, self-organizing neural networks, to the automation of the second of these problems, i.e., given the number of nodes and the mesh's topology, deciding how to place the mesh on the domain in such a way as to optimize the productivity of the finite-element method. (For work concerning the third point, i.e. efficient numbering of the nodes, see ref. 5.)

The density of the mesh affects the accuracy of the finite-element results. A finer mesh would give more accurate solutions but also would necessitate a larger computational effort. Thus the actual density of the mesh used in a certain computation is a compromise between accuracy and cost. The main parameter that controls the density of the mesh is called the *mesh parameter*; this is roughly the size of the largest element in the mesh. Of course, the density of the mesh should not necessarily be uniform. The mesh may be finer in some regions and coarser in others.

The problem of generating and placing a mesh, say, in two dimensions, is not merely a problem of dividing a given area into nonoverlapping triangles and/or quadrilaterals of a given maximum size. This is so because finite-element meshes must have certain properties in order to be acceptable for computation. The following guidelines are considered standard. In stating them, we refer to the two-dimensional case for simplicity.

1. The mesh should be finer in regions where the solution is believed to be changing rapidly or to have large gradients. Thus smaller elements should be used near singularity points such as reentrant corners or cracks, near holes, near small features of the boundary, near the location of rapidly changing boundary data, at and near inhomogeneities, etc.
2. All elements should be well proportioned. The aspect ratio of the element (namely, the ratio between its largest and smallest dimensions) should be close to unity. Square elements are the best quadrilaterals, but even an aspect ratio of 1.5 or 2 is acceptable.
3. All interior angles of the element must be significantly smaller than 180 degrees. For example, a quadrilateral with three of its vertices lying on a nearly straight line is usually unacceptable.
4. Transition from large elements to small elements must be made gradually. The ratio between the sizes of two neighboring elements may be 1.5 or 2 but should not be much greater than this.

In the early days of the FEM (in the sixties and early seventies), finite-element meshes were produced manually. This was a tedious task that also easily admitted errors in the data description. As the method was applied to successively larger problems, time for mesh preparation also became prohibitive. These difficulties have been alleviated by the development of automatic mesh-generation algorithms.

Currently, there are several schemes in use for automatic mesh generation. One major class of schemes is based on conformal mapping. See, for example, refs. 8, 11, and 14. Here, a regular mesh in a simple domain (e.g., a rectangle) is mapped into the actual domain under consideration using numeric conformal mapping. This procedure can produce high-quality meshes but is sometimes expensive. The high computational cost of this method is due to the fact that it requires the solution of Laplace's equation (or some equivalent computational effort) to generate the mesh prior to the solution of the partial differential equation under consideration.

Other two-dimensional schemes are triangulation schemes, which produce, by some construction, meshes made of triangles. See, for example, refs. 1 and 12 through 15.

Although it is well known that quadrilaterals usually perform better than triangular elements of the same order,[7] it is much easier in general to generate an acceptable triangulation than an acceptable mesh composed of quadrilaterals. Three-dimensional mesh generation is yet much more complicated. Because of this, for the three-dimensional case, tetrahedral schemes completely dominate, although it is known that they are not always a good choice for elements, in terms of accuracy.

A review and discussion of various mesh generation methods can be found in two recent books.[4,9]

2 METHODS OF THIS PAPER

Our approach has been to split this rather complicated global optimization problem into several parts. On the one hand, there are the decisions regarding the size of the mesh, the kinds of elements, and the appropriate densities in different regions of the domain, while on the other hand, one has to realize the mesh by specific assignments of geometric coordinates to the nodes. We anticipate the first part being accomplished by an expert system(s) that will, based on geometric and physical considerations, decide on the regions of interest and desired density distribution of the elements; the second part (which is the work presented here) is realized by self-organizing neural networks.[10]

A self-organizing neural network, as described, for example, by Kohonen,[10] is a system of neurons linked by a topology. Such a network can then learn to adjust its weight parameters, based on the input, in such a way as to automatically create a map of responsive neurons that topologically resembles the input data. The map so generated should in principle

automatically favor most of the heuristic rules stated above, and so the map can be taken as the placement of the mesh.

The methods presented here are independent of the specific topology chosen for the mesh. While the meshes we have used in our experiments have been chosen to consist of quadrilaterals, in principle, our methods will work for any mesh, even mixed triangular and quadrilateral ones. (As stated earlier, while it is known that quadrilateral meshes generally give better results than triangular ones,[7] the accepted methods of placing the mesh are more developed in the case of triangles.[14])

To evaluate our methods, we compared our results with a fully developed automatic mesh generator (for triangles) PLTMG[1] by comparing the results in solving a series of boundary value problems over a two-dimensional domain.

The appropriate placement of the mesh has as a heuristic component the choice of the *density function* that expresses which part of the domain should be approximated more closely than others (typically the density is higher near corners or other interesting geometric phenomena where the linear approximation inside an element is intrinsically worse). Note that the best density choice may actually depend on the solution to the differential equation. Nonetheless, in many cases qualitative information is available prior to the solution of the equation (e.g., from the boundary conditions) so that it is not unreasonable for a system to generate an appropriate density function based on the statement of the partial differential equation problem and boundary conditions. For example, it is expected that the solution exhibits high gradients near a sharp corner in the boundary. In such cases, a density function may be chosen to exploit this information. If nothing is known a priori about the solution, a density function may still be chosen in an adaptive a posteriori fashion. For example, the problem may be solved preliminarily with a uniform-density mesh (perhaps with relatively few nodes), and then a density function may be constructed based on the gradients of this solution prior to resolving it. (In our examples we chose the density function by hand, with the knowledge of the available exact solutions.)

Mesh quality can be judged by eye in the two-dimensional case. However, we also found it necessary to define an analytic measure of the quality. Below we describe this measure of mesh quality; essentially it is a mathematical realization of the preceding heuristic rules on the way a mesh should vary to obtain good numeric results. Currently, we use visual quality to decide when to cease improving the mesh, but this can be replaced by examining the changes in mesh quality function. An analogue of this quality measure can be developed for higher-dimensional meshes.

As stated, the essence of our implementation is the Kohonen self-organizing neural network algorithm.[10] This algorithm allows a network to choose its weights in such a way as to fix its topological elements in *weight space* in such a way as to mimic as closely as possible the arrangement of sample input data. In other words, the neural network becomes a representative map of the sample data information. This is exploited by us, in order to arrange for the placement of the finite-element mesh, by identifying the mesh nodes with neural nodes and identifying the weight space with the physical space of the domain, thereby causing the network to be an approximation of the density function. This happens by randomly choosing sample points of the domain as input to the self-organizing neural network in direct correspondence to the density function (see below for further details).

Thus the "coordinization" of the mesh is carried out automatically by the Kohonen algorithm, with the only input necessary being sample points of the domain chosen randomly to reflect the desired density function. The mesh then "self-organizes" to make the best possible representation of the domain by the mesh elements.

It turns out that such a straightforward procedure has some difficulties in our context, which required some sophistications and modifications when adapting the algorithm. That is,

1. A finite-element mesh has to fit exactly inside the domain and reach the boundaries.
2. Computational requirements are somewhat high.
3. Nonconvex domains require special techniques.

Our implementation dealt with these problems with the following techniques:

1. The algorithm actually uses an interweaving of several Kohonen algorithms. There is an interweaving of a two-dimensional Kohonen algorithm on the mesh with a one-dimensional Kohonen algorithm on each connected component of the boundary. More will be stated in the sequel, but note that this suggests the correct generalization to higher dimensions.
2. We used an adaptation to the Kohonen algorithm suggested by Tzvi and Iaakov[16] that resulted in an increase in speed of around 75% without degradation of performance.
3. A special modification to the algorithm was made to handle nonconvex domains properly. This algorithm is not as advanced as the convex one, but results are already comparable with the PLTMG standard.

3 THE NEURAL NETWORK ALGORITHMS

3.1 The basic Kohonen algorithm and the Tzvi-Iaacov speedup

Complete details of the Kohonen algorithm can be found in ref. 10 or ref. 6. As stated earlier, the Kohonen algorithm is designed to allow a given set of neurons organized with a topology to "self-organize" itself in such a way as to make

(1) each neuron equiprobabilistically likely to respond to an impulse in the data set and (2) the topology preserved in the sense that nearby neurons will respond to nearby impulses.

In our context, this algorithm works as follows:

1. A particular topology of nodes (or neurons) is chosen. Each node is associated with a location in a space (e.g., by cartesian coordinates). In neural network terminology, the weight space of the neuron is associated with the cartesian coordinates.
2. Input is a sequence of points in the space, chosen randomly according to the desired distribution.
3. For each input, a point in the topology is selected by a form of "winner-takes-all" competition [for example, (roughly speaking) the closest (in the current coordinates of the nodes)].
4. The location of the chosen node is adjusted, as is the coordinates of all nodes within a certain neighborhood of this chosen node. The adjustment is a movement of the chosen node toward the input point. The amount of the movement is determined by a parameter α that decreases dynamically as the algorithm proceeds in time (see Figure 1 and Figure 2).

If the input points form a faithful representation of the domain, then the nodes eventually form a map approximating the given domain. Note, however, that because of the probabilistic nature of this scheme, meshes generated for symmetric regions will not be perfectly symmetric.

This general algorithm is independent of the dimensionality. What is needed is a representation of the nodes in a space and an appropriate generator of input points.

Tzvi and Iaakov[16] suggested an adaptation of this basic algorithm that speeds things up substantially. Essentially (see refs. 16 and 17 for details), the network is broken into subregions, and each region is represented by one member, roughly from the center of the subregion. Then the winner-takes-all competition proceeds first by competition only among the representatives and then among the members of the chosen region (see Figure 3).

We found that the use of one level of this adaptation resulted in a 75% improvement in the speed of the Kohonen algorithm without any change in quality of results. (In principle, this algorithm can work with several levels of representation, but we did not need this.)

3.2 Adaptation to our problem

3.2.1 *Density functions*

In principle, then, one needs only the choice of a network and a density function. For example, a uniform density function is easily generated by first enclosing the body inside a circumscribing rectangle and choosing an x and y coordinate by one of the standard random number generators (we used *random* from the standard *C* library package) and then rejecting any points that fall outside the body itself.

One then simply uses the randomly chosen points in the region as input to the Kohonen algorithm network. In the same way, any computable density function on the domain can be input to the Kohonen network.

We implemented nonuniform densities by taking the composition of squares of uniform ones centered at various "hot spots." In our system, a user indicates areas of interest with a pointing device (a graphic mouse), and then the appropriate density functions are generated automatically as the square of the uniform density function on [0, 1] with (0, 0) identified with the hot spot (see Figure 4). We point out that any other method of generating the density function is acceptable. Note that often in solving partial differential equations with boundary conditions, if the submitted data (e.g., boundary condition, geometry, load f, etc.) exhibit some nonsmooth behavior, as in the case of cracks, corners, or concentrated loads, this gives a hint as to what the density function should look like. In an elliptic partial differential equation, these are the only cases where a nonuniform density is needed, and so we have a good idea in advance of solution where the hot spots are. On the other hand, for hyperbolic partial differential equations, the solution may be nonsmooth even if all the data are smooth, as in the appearance of shocks in fluid-flow problems. In such cases, it is more complicated to predict in advance where the hot spots are, but one can resort to adaptive methods (e.g., solving first with a sparser uniform mesh and then resolving with a new mesh relating to the less accurate solution).

3.2.2 *Measures of mesh quality*

To test the current version of the algorithm, we used two measures of success. First, we used the following as our measure of the quality of a mesh, reflecting the heuristic rules given in Section 1 (here for a given element a^e refers to the largest side of the quadrilateral and b^e refers to the smallest: $E_1^e = 1 - b^e/a^e$ giving a measure of the aspect ratio, $E_2^e = \max_{i=1}^4 |1 - \frac{2}{\pi}\text{angle}_i|$ measuring how close all the quadrilateral angles are to 90 degrees, and $E_3^e = \max_{\text{neighbors}} |1 - \min_{n=1}^{N^e}\{a^e/a_n^e, a_n^e/a^e\}|$ measuring how similar an element is to its neighboring ones.) Then, allowing different positive weights w_i to the different measures, we have

$$\text{Quality(mesh)} = \sum_{\text{elements}} w_1 E_1^e + w_2 E_2^e + w_3 E_3^e$$

(In the work presented here, lacking any further information, all the w_i values are 1.)

This function allows one to measure how various changes in the algorithm result in improvements in the mesh. (The smaller the value, the better is the mesh.) This allows one to measure quantitatively how the mesh changes as the algorithm proceeds. Because of the stochastic nature of the algorithm, in principle, random effects can cause this not to be monotone (i.e., the net can get worse); over many itera-

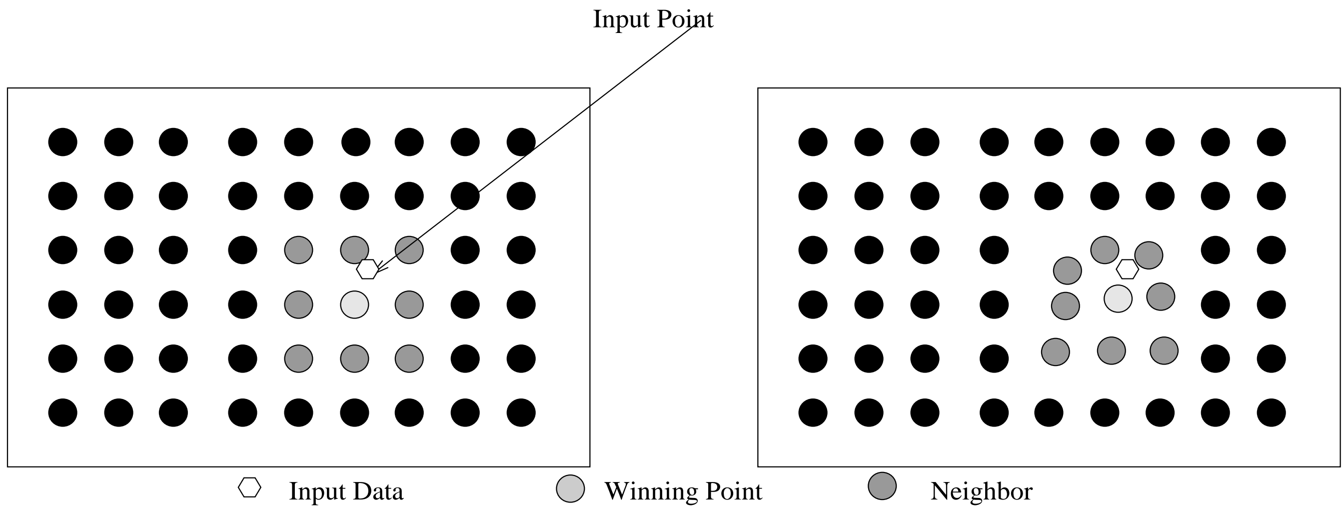


Fig. 1. An input point is chosen according to the chosen density function. A winning node is chosen, and then the positions of the winner and its neighbors are adjusted toward the input point.

Iteration 0; Initial Setup	Iteration 500; Quality = 288.10	Iteration 2000; Quality = 237.78	Iteration 4500; Quality = 226.00
	Iteration 6000 Quality = 222.81	Iteration 12000 Quality = 207.79	Iteration 30000 Quality = 202.46

Fig. 2. A sequence of snapshots of a mesh.

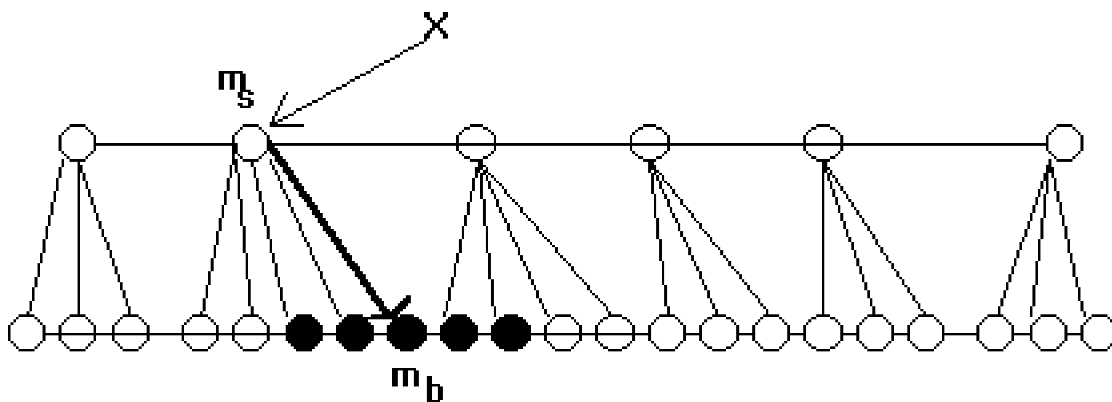


Fig. 3. The organization of the Tzvi-Iaakov speedup.

tions it eventually always decreases, and this is born out by our experiments. In a fully automated system, this measure could be used as an indication of when to halt the algorithm; in our experiments, the halting was always done manually.

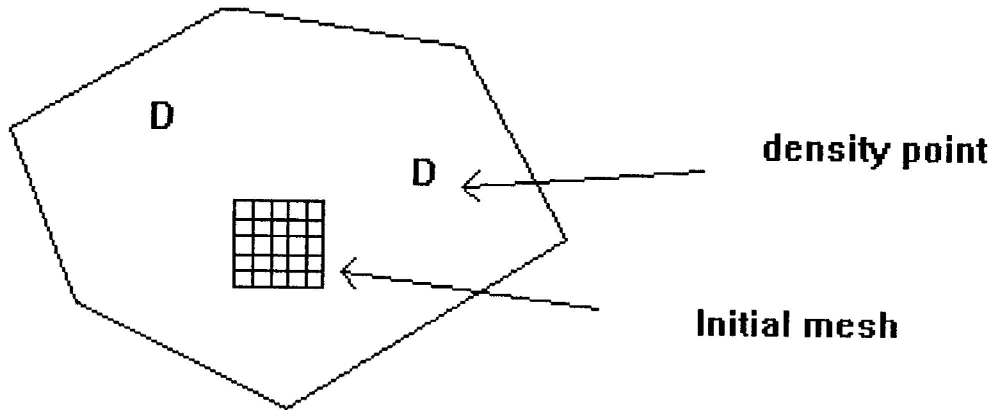


Fig. 4. A domain with the initial mesh and two high-concentration "hot spots" designated by D.

(For each experiment, we indicate the number of iterations.)

In order to compare the quality of the resulting mesh externally, we decided to send the output of our algorithm directly to an FEM partial differential equation solver and to compare the solutions of various partial differential equations with boundary conditions with those run on a popular professional mesh generator PLTMG.[1] It is not possible to compare the meshes directly because PLTMG generates triangles and our examples used quadrilaterals.

Actually, PLTMG also includes an FEM partial differential equation solver, but to keep the comparison fair, we only used PLTMG as a mesh generator and gave its mesh as an input to the same FEM solver.

Accordingly, we compared the quality of solutions of partial differential equations with boundary conditions as solved using the different meshes. We used equations with analytic solutions so we could measure the precise error in the solutions. Our examples were of the form $u_{xx}+u_{yy}+f(x, y) = 0$. (The boundary conditions were given by evaluating the solution at the boundary points.) In all cases, we report both the average error per node and the error per value of the solution function. (Here u is the analytic solution and u_h is the numerically computed solution.)

$$\text{Error/node} = \frac{\sum_{\text{nodes}} |u(\text{node}) - u_h(\text{node})|}{\#(\text{nodes})}$$

$$\text{Error/value} = \frac{\sum_{\text{nodes}} |u(\text{node}) - u_h(\text{node})|}{\sum_{\text{nodes}} |u(\text{node})|}$$

3.2.3 *Covering the domain*

For our application, it is important that eventually the network completely cover the domain, i.e., that boundary points of the network fall on boundaries of the domain. The basic Kohonen map does not typically fulfill this constraint. For example, for the body in Fig. 5, the result is as in Fig. 6.

This constraint resulted in the implementation of some heuristic rules and the interweaving of several Kohonen algorithms together, which we now proceed to describe. For

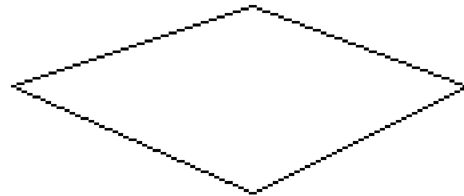


Fig. 5. A sample body.

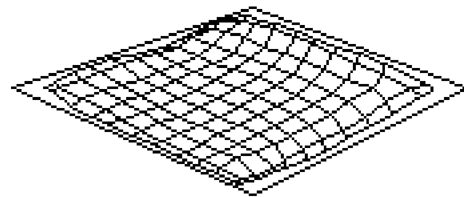


Fig. 6. The result of a basic Kohonen map on Fig. 5. Notice that the boundary is not reached.

example, it is possible to guarantee reaching the boundary by simple numeric rounding *provided sufficiently frequent sample points occur on the boundary*. However, it is highly unlikely for such points to be chosen randomly. In an earlier version, we did this heuristically simply by choosing roughly one of every seven points to be on the boundary (after an initial period where the points where chosen uniformly throughout the domain). This, however, leads to a distortion of the distribution about the boundary that can be seen visually in Fig. 7.

3.2.4 *Dimension* 1 *Kohonen and interweaving*

In order to compensate for this affect, a secondary one-dimensional Kohonen network is used to adjust the boundary; the two Kohonen algorithms are then interwoven in parallel.

However, this interweaving is a little bit complicated. First, note that the boundary is a one-dimensional simplex in our examples. (In principle, there might be more than one such sim-

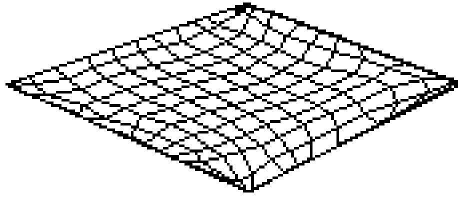


Fig. 7. The result of a Kohonen map obtained by forcing points to be on the boundary of Fig. 5. Notice the distortion about the boundary.

Table 1
Tables of values of α and α'

Iterations	Value of α
1–9	0.8
10–999	0.3
1000–1999	0.2
2000–3000	0.1
3001–7000	0.05
7001–8000	0.08
8001–10,000	0.01
Over 10,000	0.001

Iterations	Value of α'
3000–3999	0.45
4000–4999	0.35
5000–6000	0.25
6001–10,000	0.10
10,001–11,000	0.11
11,001–13,000	0.03
Over 10,000	0.007

plex, i.e., one for each connected component of the boundary. This is handled simply by using several of these algorithms in parallel.)

In order to distribute the points appropriately on the boundary (but after the net has reached the boundary), we apply a Kohonen algorithm to this one-dimensional simplex. That is, we start with a separate Kohonen adjustment parameter α' and a neighborhood *in the boundary simplex*. In practice, the boundary radius was taken to be 1, and then it was decreased to 0 (i.e., only the chosen node is moved). (See Table 1 for the changes of these parameters as a function of the number of iterations.)

Thus, for this algorithm, one choses randomly points on the boundary and applies this one-dimensional version to the one-dimensional simplex. However, in parallel, one continues to update the two-dimensional simplex.

In addition, once a point is on the boundary, we do not allow it to leave the boundary (as might occur from the two-dimensional algorithm); in effect, the two-dimensional algorithm now acts only on the interior of the original net.

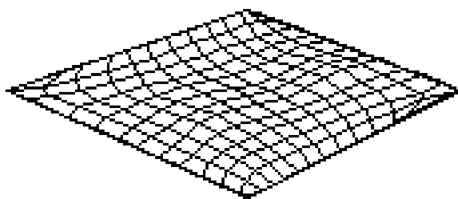


Fig. 8. The result of the interwoven Kohonen map on Figure 5.

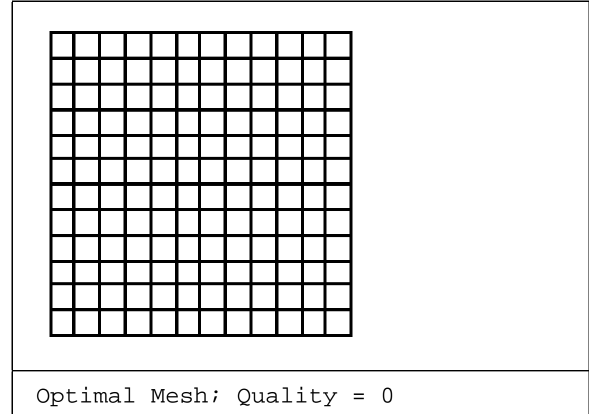


Fig. 9. An optimal mesh on a square domain.

Note that the preceding requires a balancing of the effects. We do this* by

1. Running a "pure" Kohonen algorithm on the domain for around 1500 iterations. Boundaries are typically not reached during this stage.
2. Between 1500 to 3000 iterations running a "distorted" Kohonen algorithm by choosing one of every seven sample points to be on the boundary of the domain. (Actually, the algorithm would alternate chosing 30 interior points and then 5 boundary points.)
3. Between 3000 to 5000 iterations running both the one-dimensional and the two-dimensional Kohonen maps, again with a ratio of one out of every seven points chosen on the boundary.
4. Over 5000 iterations running both the one-dimensional and the two-dimensional Kohonen maps. However, at this point, points on the boundary can no longer be moved into the interior. This effectively means that the two-dimensional Kohonen map is now running only on the original mesh minus its boundary.
5. The one-dimensional Kohonen parameter α' is adapted independently of the two-dimensional Kohonen parameter α. See Table 1 for the values per iteration actually used for these parameters.

Figure 2 shows a sequence of "snapshots" over different numbers of iterations of the development of a mesh.

Figure 8 shows the result on the example in Fig. 5 with the interwoven algorithm.

In terms of our quality function, the interwoven algorithm produces a mesh with value 165.050814; the previous basic

* Both the specific choices of numbers of iterations and "cooling schedules" for α and α' were found by experimentation and so are strictly valid only for the examples in this paper. However, they give rough guidelines for other instances.

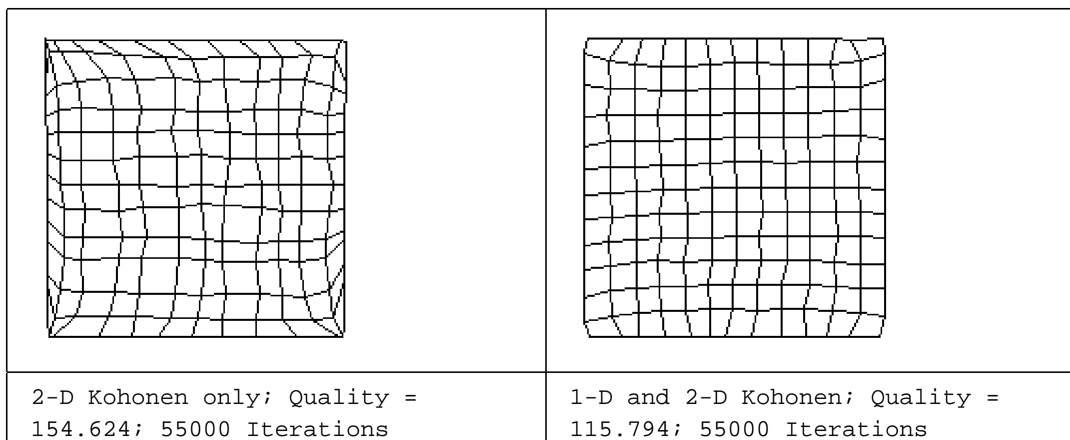


Fig. 10. Comparison of neural net–generated meshes, two-dimensional and interwoven one- and two-dimensional versions.

Table 2
Table of errors arising when applying the FEM on an optimal mesh

Rectangular domain optimal (169 *nodes*, 144 *elements*)			
$u(x, y)$	$f(x, y)$	*Error/node*	*Error/value*
$x^3 + y^3$	$-6x - 6y$	2.463314E-04	8.301765E-06
$\sin x + \sin y$	$\sin x + \sin y$	1.314296E-04	1.103902E-04

Table 3
Table of errors arising on applying FEM to neural network–generated meshes

Rectangular domain (169 *nodes*, 144 *elements*)					
		Error/node		*Error/value*	
$u(x, y)$	$f(x, y)$	2-*D only*	1-*D and* 2-*D*	2-*D only*	1-*D and* 2-*D*
$x^3 + y^3$	$-6x - 6y$	2.688617E-03	7.635503E-04	8.900321E-05	2.522708E-05
$\sin x + \sin y$	$\sin x + \sin y$	2.832740E-04	2.279894E-04	2.082413E-04	1.954924E-04

Kohonen map produces a mesh with the worse-quality value 171.255656.

It is interesting to compare these results on the square with a uniform density where the optimal setting is known. That is, a square with 169 nodes will optimally look like Figure 9.

In contrast, Fig. 10 shows the meshes generated by the "simple" two-dimensional Kohonen algorithm and by the interwoven one.

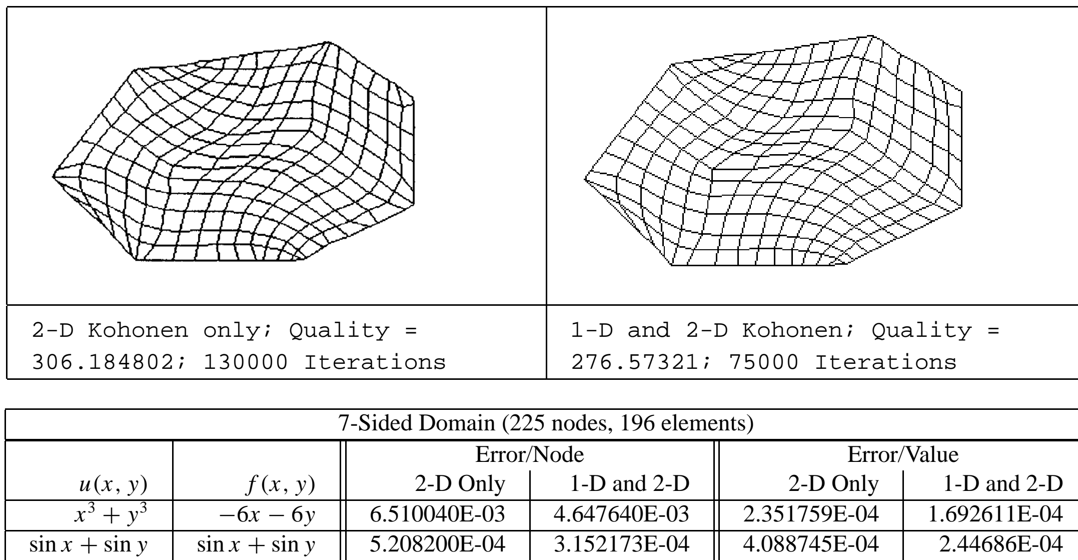
One also can compare the results of solving boundary value problems on these different meshes. Table 2 gives the results for two choices of partial differential equation boundary value problems for the optimal mesh, while Table 3 gives the corresponding results generated by the "simple" two-dimensional algorithm and by the interwoven one.

In Figs. 11–12 we display the comparison on other bodies listing as well their results for the solutions of various partial differential equations. In all cases, the interwoven algorithm is superior to the two-dimensional algorithm.

3.2.5 *Nonconvex domains*

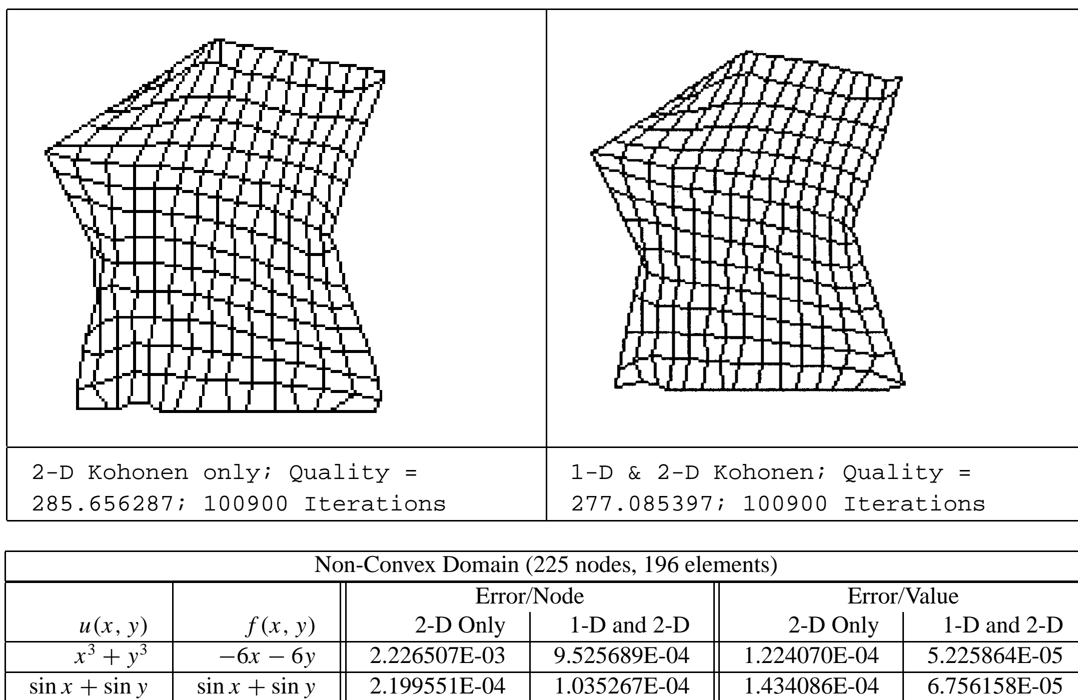
Nonconvex domains have special problems. A direct use of the Kohonen algorithm would cause elements and even nodes to migrate over the boundary lines (see Fig. 13).

To avoid this, what is needed is a different metric appropriate to the body that causes nodes and elements to stay within the body. One can conceive of such a metric being formed by a composition with the usual metric and a conformal mapping. However, as an experiment, we proceeded with a more primitive algorithm. We simply discarded any sample points that resulted in an "illegal" motion of the nodes (see Fig. 14).



7-Sided Domain (225 nodes, 196 elements)					
		Error/Node		Error/Value	
$u(x, y)$	$f(x, y)$	2-D Only	1-D and 2-D	2-D Only	1-D and 2-D
$x^3 + y^3$	$-6x - 6y$	6.510040E-03	4.647640E-03	2.351759E-04	1.692611E-04
$\sin x + \sin y$	$\sin x + \sin y$	5.208200E-04	3.152173E-04	4.088745E-04	2.44686E-04

Fig. 11. Comparison of neural net–generated meshes, two-dimensional and interwoven one- and two-dimensional versions.



Non-Convex Domain (225 nodes, 196 elements)					
		Error/Node		Error/Value	
$u(x, y)$	$f(x, y)$	2-D Only	1-D and 2-D	2-D Only	1-D and 2-D
$x^3 + y^3$	$-6x - 6y$	2.226507E-03	9.525689E-04	1.224070E-04	5.225864E-05
$\sin x + \sin y$	$\sin x + \sin y$	2.199551E-04	1.035267E-04	1.434086E-04	6.756158E-05

Fig. 12. Comparison of neural net generated meshes, two-dimensional and interwoven one- and two-dimensional versions.

This solves the problem of crossing the boundary but at the expense of (1) an increase in computation time and (2) distortion of the density function. Because of (2), we were not expecting particularly good results. In our experiments, however, while the results on the nonconvex domain are definitely inferior to those on the convex domains, nonetheless, the results are comparable with the results given by PLTMG. This indicates the robustness of our method and the superiority of working with quadrilaterals over triangles.

Nonetheless, it is clear that this algorithm can and should

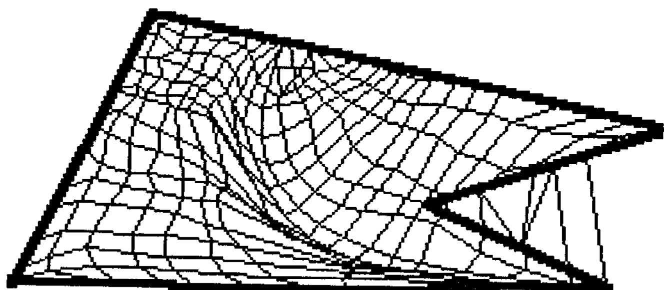


Fig. 13. A direct use of Kohonen maps over a nonconvex domain. Note that elements have crossed the boundary lines.

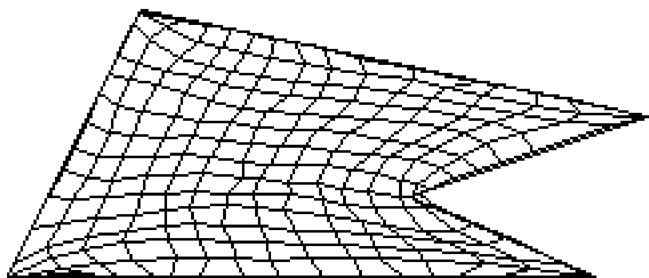


Fig. 14. The result of discarding sampled points that cause "illegal" motion (see text).

be improved. For example, we could not expect it to work in a figure with a "bottleneck."

4 DESCRIPTIONS, TABLES OF TESTS, AND DISCUSSION

In order to compare the quality of the resulting mesh externally, we decided to send the output of our algorithm directly to an FEM partial differential equation solver and to compare the solutions of various partial differential equations with boundary conditions with those run on a popular professional mesh generator PLTMG. It is not possible to compare the meshes directly because PLTMG generates triangles and our examples used quadrilaterals. In all cases we tried to compare PLTMG and neural network (NN) meshes with a similar number of nodes.*

Actually, PLTMG also includes an FEM partial differential equation solver; however, to keep the comparison fair, we only used PLTMG as a mesh generator and gave its mesh as an input to the same FEM solver.

That is, we took the following:

- A set of domains, all either convex or "near-convex" and nonconvex.
- Three different boundary value problems on these domains. Exact solutions are known for these problems. Thus we could calculate the actual error at each node.
- Different densities. For the problem with the exponential solution, it is desired to have a nonuniform density. For each of the domains we solved this problem with both uniform and nonuniform density using the NN.

For each generated mesh and problem we input it into a FEM solver and then computed the quality of the solution using our measures for error/node and error/value.

We solved problems of the form $u_{xx}+u_{yy}+f(x, y) = 0$. In each case we chose $u(x, y)$ and then found the corresponding boundary conditions and function $f(x, y)$ such that $u(x, y)$ is the exact solution of the problem. In this manner, we easily synthesized exact solutions and were thus able to measure the error directly.

Figures 15 to 26 present the results of experiments. There are three figures for each of the four combinations of uniform/nonuniform density and convex/nonconvex domains.

In order to summarize these, we have prepared two tables (see Tables 4 and 5). Table 4 lists for each experiment and for each of the two measures which method, PLTMG or NN, gave a better result. Table 5 replaces "better" by "=" for cases where the difference was less than an order of magnitude. Looking over the results, we note the following points:

- PLTMG and NN produce results of roughly comparable quality, with NN superior overall.
- NN has an advantage where one wants to have nonuniform density functions. This can be seen clearly, for instance, in the examples of convex nonuniform density, where PLTMG was better in error/node but worse in error/value. This means that NN placed its resources where needed. This also can be seen visually; the network placed

* Note that part of the advantage of the method of this paper is in fact its ability to use quadrilaterals. If one takes the resulting quadrilateral mesh and simply bisects each quadrilateral to produce a triangular mesh, the quality can vary substantially depending on the specific mesh. For example, when, at the suggestion of the referee, we did this for the first partial differential equation of Fig. 17, the results (error/node 2.788065E-03; error/value 9.211537E-05), while (as expected) worse than the NN, were still better than PLTMG. However, this is probably a consequence of the symmetry of the mesh. In general, if a quadrilateral element is far from being optimal, then the resulting triangular elements can be poor, even close to degenerate elements. It is also possible to apply our method to a given triangular mesh directly, i.e., to apply the neural network algorithm to a topology of triangles. While we have not done this systematically, a simple experiment using our method for a triangular mesh for the same partial differential equation did indeed give the improved results (error/node 1.745574E-03; error/value 5.829219E-05) while, as expected, still falling between PLTMG and the NN.

Table 4
Best results

Figure	*Density*	*Convex*	*Best E/N*	*Best E/V*
Fig. 15	u	c	P/P	P/P
Fig. 16	u	c	N/P	N/P
Fig. 17	u	c	N/N	N/N
Fig. 18	u	n	N/P	N/P
Fig. 19	u	n	P/N	P/N
Fig. 20	u	n	P/P	P/P
Fig. 21	n	c	P	N
Fig. 22	n	c	P	N
Fig. 23	n	c	P	N
Fig. 24	n	n	N	N
Fig. 25	n	n	N	N
Fig. 26	n	n	N	N

Note: This table shows which mesh generator, P (PLTMG) or N (neural network), gave better results based on both the error/node and error/value. Densities are either uniform "u" or not "n"; experiments were either convex "c" or nonconvex "n."

Table 5
Order of magnitudes distinguishability

Figure	*Density*	*Convex*	*Best E/N*	*Best E/V*
Fig. 15	u	c	=/=	=/=
Fig. 16	u	c	=/=	=/=
Fig. 17	u	c	N/=	N/=
Fig. 18	u	n	N/=	=/=
Fig. 19	u	n	=/=	=/N
Fig. 20	u	n	P/P	=/=
Fig. 21	n	c	=	=
Fig. 22	n	c	=	=
Fig. 23	n	c	=	=
Fig. 24	n	n	N	N
Fig. 25	n	n	N	N
Fig. 26	n	n	N	=

Note: This table shows which generator is better only when one was an order of magnitude better, otherwise designating the entry as =. Two entries in a position refer to two different boundary value problems as described in the appropriate figure. Densities are either uniform "u" or not "n"; experiments were either convex "c" or nonconvex "n".

by NN is concentrated around the maximal point of $u(x, y)$ and is roughly symmetric. This was even the case in nonconvex domains.

- If chosen properly, NN produces better results with a nonuniform density function than without one. Even with uniform density, it competes successfully with PLTMG.
- The interwoven algorithm combining one-dimensional and two-dimensional Kohonen maps is definitely superior to using a two-dimensional Kohonen map.
- NN produces reasonable answers even on nonconvex domains, but the algorithm cannot compete with the professional ones on all domains.

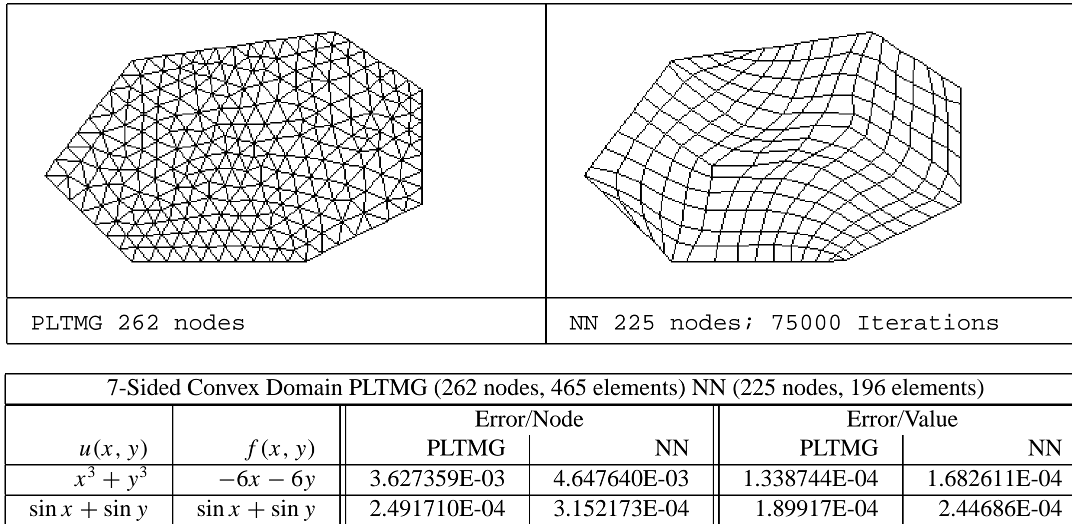
5 FUTURE WORK AND CONCLUSIONS

These results are quite encouraging because there are many techniques included in PLTMG that are not yet included in NN. In particular, PLTMG's network is determined dynamically, whereas NN's is a fixed topology. (This is why the exact number of nodes in PLTMG cannot be controlled exactly.) This results necessarily in some poor elements.

- A future goal is to add some ability to adjust the topology (i.e., by removing elements). It is possible that the algorithm presented in ref. 3 may be adapted to this case.
- In another approach to the same problem of poor elements, it should be possible to "tweak" the mesh generated by the algorithm, e.g., by an expert system to modify the worst elements as indicated by the quality function.
- An immediate extension of this work will be to allow non-simply connected domains. (All the current examples were simply connected, i.e., without "holes" in the domain, e.g., an annulus.) The main difference is that the boundary of a non-simply connected domain need not be connected; i.e., there may be several separate boundaries. The algorithm will handle this by having several one-dimensional Kohonen maps (instead of just one), one for each boundary, all simultaneously interwoven with the remaining two-dimensional map.
- Perhaps most important, it seems that the algorithm presented here is quite adaptable to higher dimensions. We anticipate a three-dimensional version working as follows:
  1. Place a three-dimensional simplex, e.g., boxes, in the center of the domain.
  2. Run a three-dimensional version of the Kohonen algorithm until the simplex reaches the two-dimensional boundaries.
  3. Then interweave two-dimensional versions of the Kohonen algorithm with the three-dimensional one until the one-dimensional boundaries are reached.
  4. Then interweave one-dimensional versions of the Kohonen algorithm on each of these one-dimensional boundaries together with the two-dimensional and the three-dimensional algorithms.

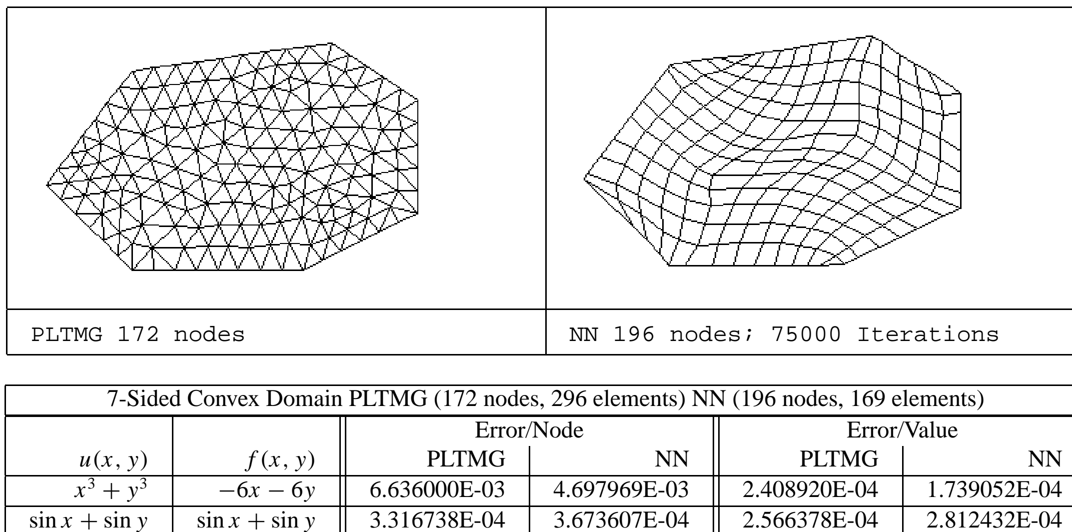
  Of course, choosing the correct mix of parameters for this three-dimensional case requires substantial experimentation.

We hope to report on progress in at least some of these directions in a future publication.



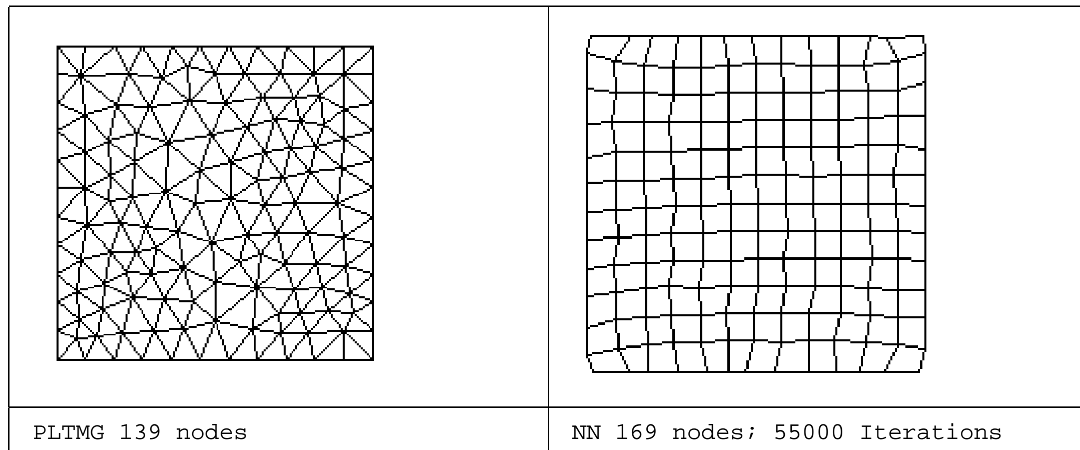
7-Sided Convex Domain PLTMG (262 nodes, 465 elements) NN (225 nodes, 196 elements)					
		Error/Node		Error/Value	
$u(x, y)$	$f(x, y)$	PLTMG	NN	PLTMG	NN
$x^3 + y^3$	$-6x - 6y$	3.627359E-03	4.647640E-03	1.338744E-04	1.682611E-04
$\sin x + \sin y$	$\sin x + \sin y$	2.491710E-04	3.152173E-04	1.89917E-04	2.44686E-04

Fig. 15. Results on a convex, uniform-density domain.



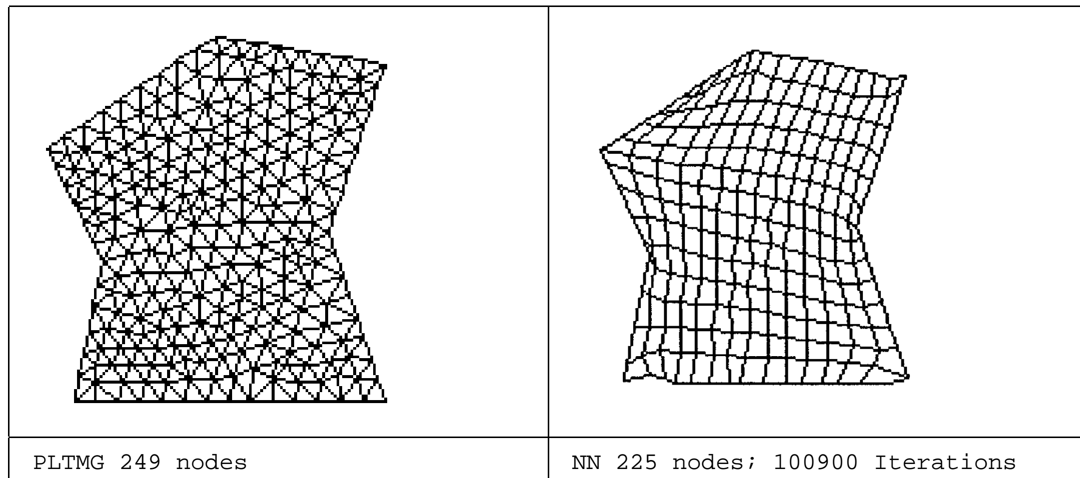
7-Sided Convex Domain PLTMG (172 nodes, 296 elements) NN (196 nodes, 169 elements)					
		Error/Node		Error/Value	
$u(x, y)$	$f(x, y)$	PLTMG	NN	PLTMG	NN
$x^3 + y^3$	$-6x - 6y$	6.636000E-03	4.697969E-03	2.408920E-04	1.739052E-04
$\sin x + \sin y$	$\sin x + \sin y$	3.316738E-04	3.673607E-04	2.566378E-04	2.812432E-04

Fig. 16. Results on a convex, uniform-density domain.



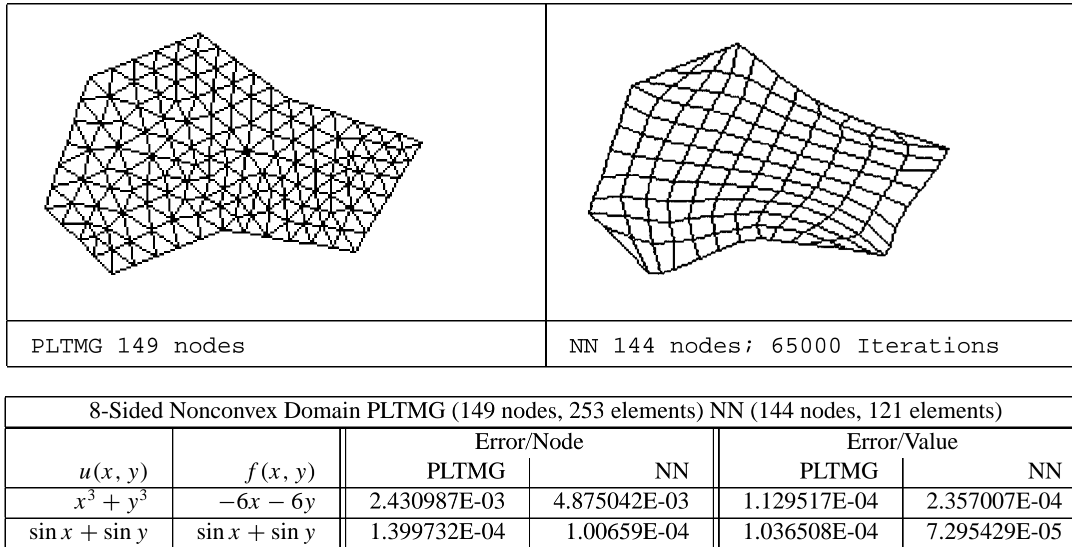
Rectangular Domain PLTMG (139 nodes, 232 elements) NN (169 nodes, 144 elements)					
		Error/Node		Error/Value	
$u(x, y)$	$f(x, y)$	PLTMG	NN	PLTMG	NN
$x^3 + y^3$	$-6x - 6y$	6.427036E-03	7.635503E-04	2.082413E-04	2.522708E-05
$\sin x + \sin y$	$\sin x + \sin y$	3.085165E-04	2.279894E-04	2.68856E-04	1.954924E-04

Fig. 17. Results on a convex, uniform-density domain.



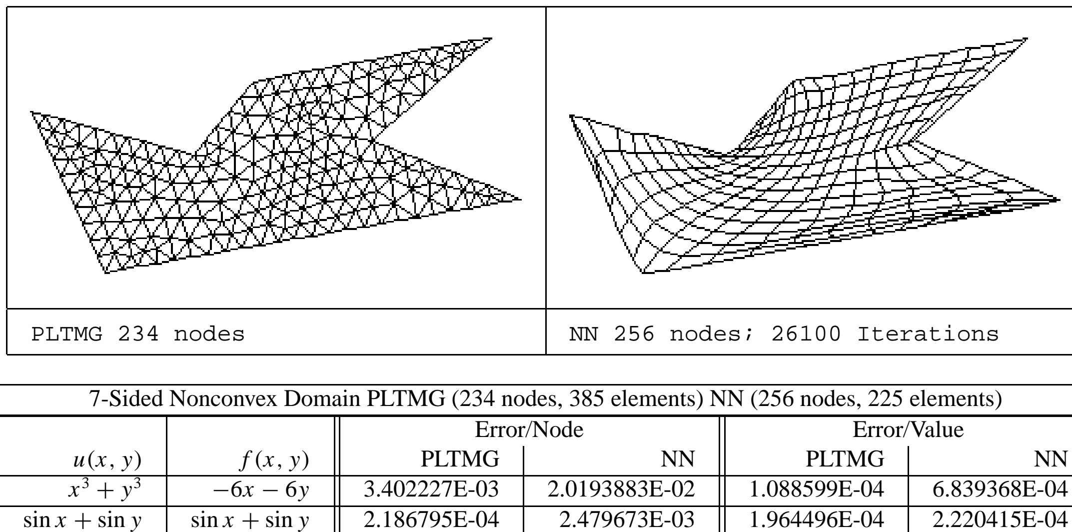
7-Sided Nonconvex Domain PLTMG (249 nodes, 437 elements) NN (225 nodes, 196 elements)					
		Error/Node		Error/Value	
$u(x, y)$	$f(x, y)$	PLTMG	NN	PLTMG	NN
$x^3 + y^3$	$-6x - 6y$	1.391667E-03	9.525689E-04	7.660676E-05	5.225864E-05
$\sin x + \sin y$	$\sin x + \sin y$	1.029398E-04	1.035267E-04	6.647789E-05	6.756158E-05

Fig. 18. Results on a nonconvex, uniform-density domain.



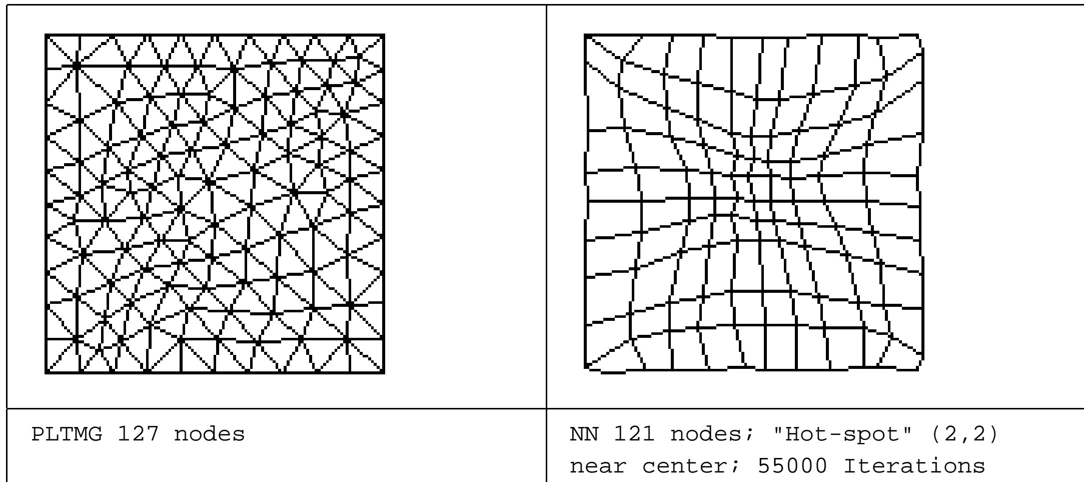
8-Sided Nonconvex Domain PLTMG (149 nodes, 253 elements) NN (144 nodes, 121 elements)					
		Error/Node		Error/Value	
$u(x, y)$	$f(x, y)$	PLTMG	NN	PLTMG	NN
$x^3 + y^3$	$-6x - 6y$	2.430987E-03	4.875042E-03	1.129517E-04	2.357007E-04
$\sin x + \sin y$	$\sin x + \sin y$	1.399732E-04	1.00659E-04	1.036508E-04	7.295429E-05

Fig. 19. Results on a nonconvex, uniform-density domain.



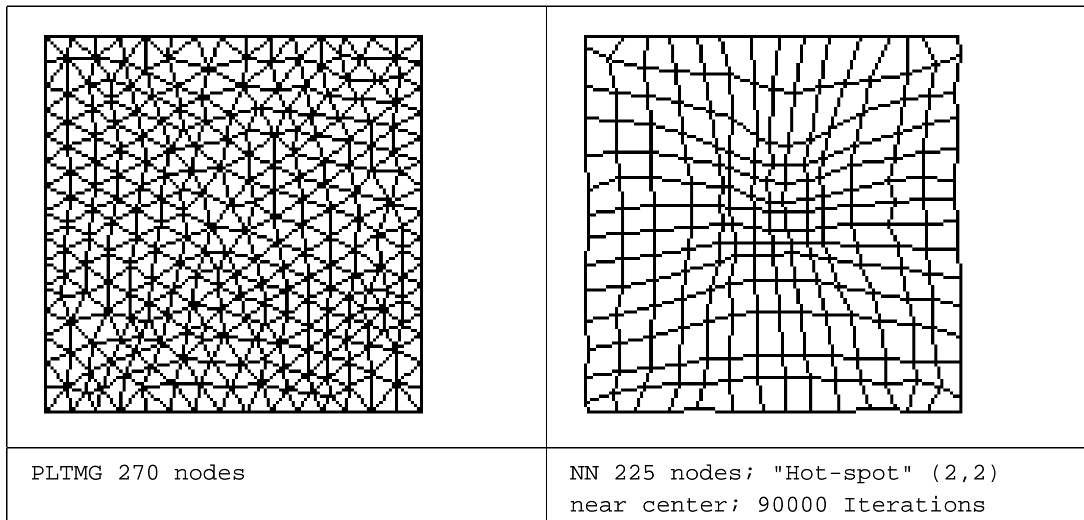
7-Sided Nonconvex Domain PLTMG (234 nodes, 385 elements) NN (256 nodes, 225 elements)					
		Error/Node		Error/Value	
$u(x, y)$	$f(x, y)$	PLTMG	NN	PLTMG	NN
$x^3 + y^3$	$-6x - 6y$	3.402227E-03	2.0193883E-02	1.088599E-04	6.839368E-04
$\sin x + \sin y$	$\sin x + \sin y$	2.186795E-04	2.479673E-03	1.964496E-04	2.220415E-04

Fig. 20. Results on a nonconvex, uniform-density domain.



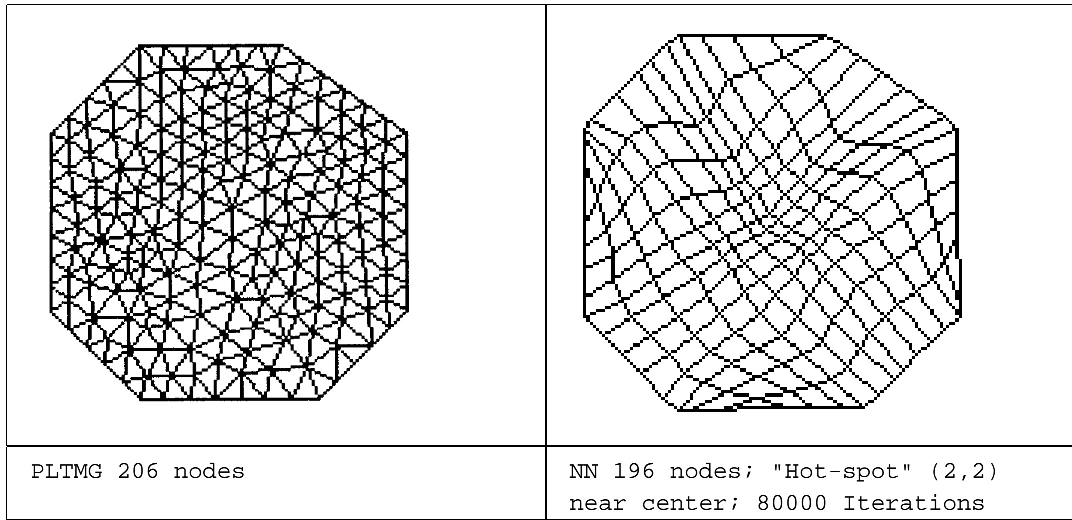
Rectangular Domain PLTMG (127 nodes, 212 elements) NN (121 nodes, 100 elements)					
		Error/Node		Error/Value	
$u(x,y)$	$f(x,y)$	PLTMG	NN	PLTMG	NN
$e^{-(x-2)^2}e^{-(y-2)^2}$	$-u_{xx}-u_{yy}$	6.244993E-02	6.780960E-02	1.226566E-01	1.202126E-01

Fig. 21. Results on a convex, non-uniform-density domain.



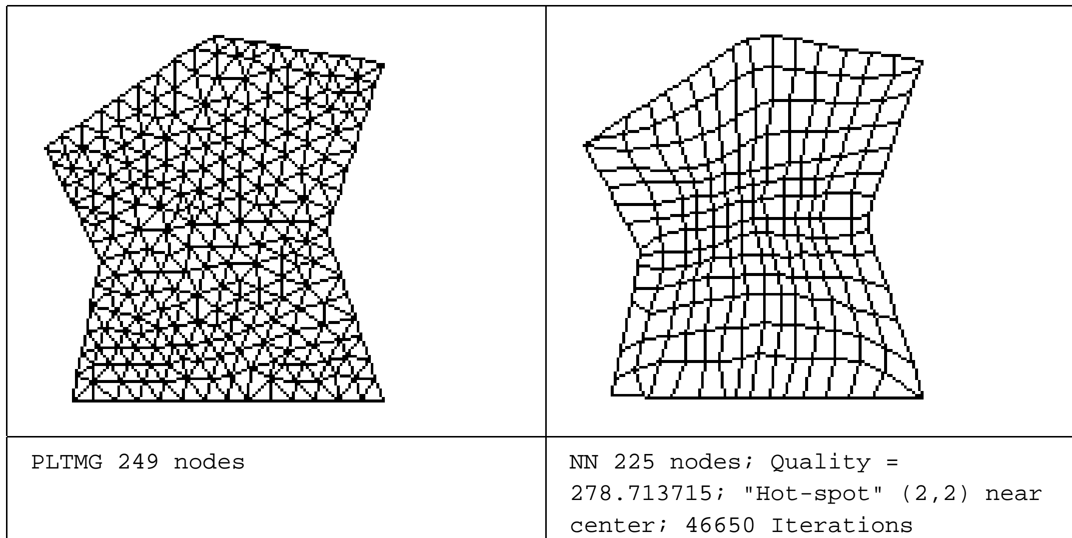
Rectangular Domain PLTMG (270 nodes, 478 elements) NN (225 nodes, 196 elements)					
		Error/Node		Error/Value	
$u(x,y)$	$f(x,y)$	PLTMG	NN	PLTMG	NN
$e^{-(x-2)^2}e^{-(y-2)^2}$	$-u_{xx}-u_{yy}$	6.841879E-02	7.193961E-02	1.311398E-01	1.256772E-01

Fig. 22. Results on a convex, non-uniform-density domain.



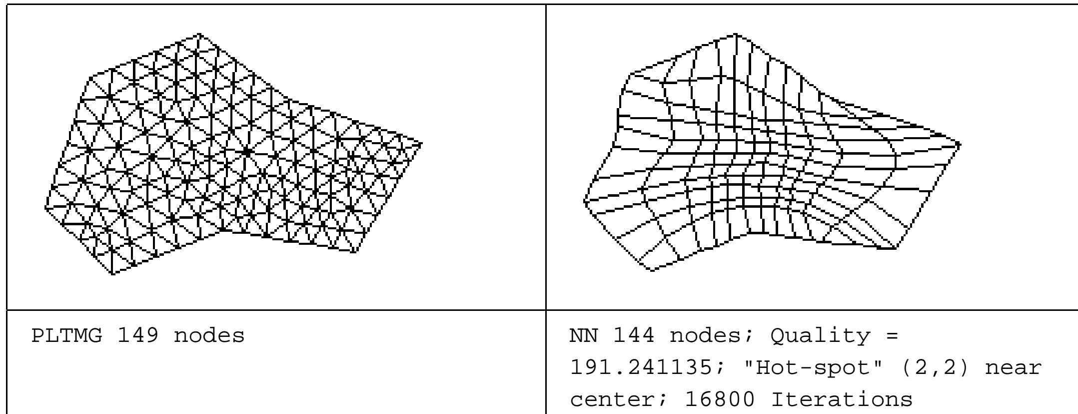
8-Sided Convex Domain PLTMG (206 nodes, 362 elements) NN (196 nodes, 169 elements)					
		Error/Node		Error/Value	
$u(x, y)$	$f(x, y)$	PLTMG	NN	PLTMG	NN
$e^{-(x-2)^2}e^{-(y-2)^2}$	$-u_{xx} - u_{yy}$	6.079690E-02	6.220065E-02	1.059934E-01	1.004030E-01

Fig. 23. Results on a convex, non-uniform-density domain.



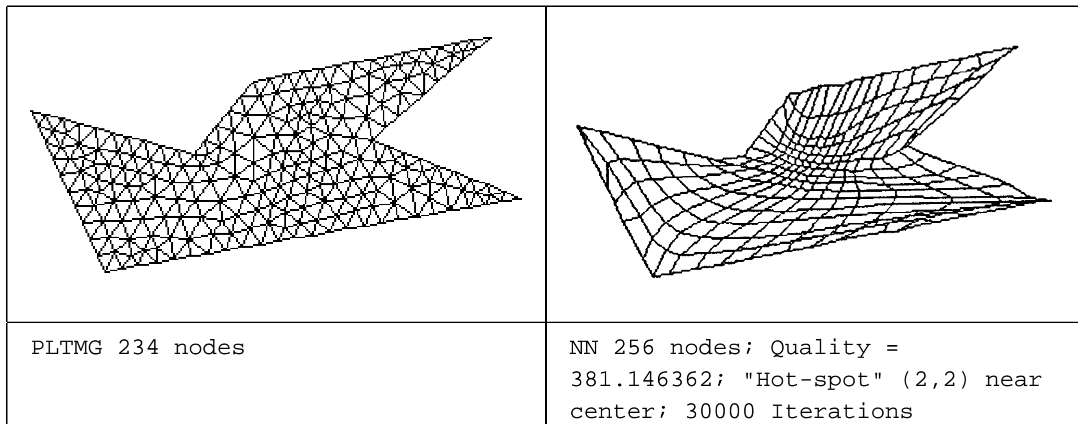
7-Sided Non-Convex Domain PLTMG (249 nodes, 437 elements) NN (225 nodes, 196 elements)					
		Error/Node		Error/Value	
$u(x, y)$	$f(x, y)$	PLTMG	NN	PLTMG	NN
$e^{-(x-2)^2}e^{-(y-2)^2}$	$-u_{xx} - u_{yy}$	2.412143E-02	7.530449E-03	4.515054E-02	9.097765E-03

Fig. 24. Results on a nonconvex, non-uniform-density domain.



9-Sided Non-Convex Domain PLTMG (149 nodes, 253 elements) NN (144 nodes, 121 elements)					
		Error/Node		Error/Value	
$u(x, y)$	$f(x, y)$	PLTMG	NN	PLTMG	NN
$e^{-(x-2)^2}e^{-(y-2)^2}$	$-u_{xx} - u_{yy}$	2.766172E-02	7.653986E-03	8.125979E-02	9.120655E-03

Fig. 25. Results on a nonconvex, non-uniform-density domain.



7-Sided Nonconvex Domain PLTMG (234 nodes, 385 elements) NN (256 nodes, 225 elements)					
		Error/Node		Error/Value	
$u(x, y)$	$f(x, y)$	PLTMG	NN	PLTMG	NN
$e^{-(x-2)^2}e^{-(y-2)^2}$	$-u_{xx} - u_{yy}$	2.864687E-02	1.811124E-03	9.151940E-02	1.159691E-03

Fig. 26. Results on a nonconvex, non-uniform-density domain.

ACKNOWLEDGMENT

Supported in part by a joint Technion–University of Haifa research grant. Some of this research was done while D. Givoli was on a sabbatical visit at Rensselaer Polytechnic Institute.

REFERENCES

1. Bank, R. E., *PLTMG*: *A Software Package for Solving Elliptic Partial Differential Equations*, SIAM Publications, Philadelphia, 1994.
2. Carey, G. F. & Oden, J. T. *Finite Elements*, vol. III: *Computational Aspects*, Prentice-Hall, Englewood Cliffs, NJ, 1984.
3. Fritzke, B. Growing cell structures: A self-organizing network for unsupervised and supervised learning, *Neural Neworks*, **7** (1994), 1441–60.
4. George, P. L., *Automatic Mesh Generation*, Wiley, Chichester, UK, 1991.
5. Manevitz, L., Givoli, D. & Margi, M., Heuristic finite element node numbering, *Computing Systems in Engineering*, **4** (1993), 159–68.
6. Hecht-Nielsen, R., *Neurocomputing*, Addison-Wesley, Reading, MA, 1991.
7. Hughes, T. J. R., *The Finite Element Method*, Prentice-Hall, Englewood Cliffs, NJ, 1987.
8. Jones, R. E., *A Self-Organizing Mesh Generation Program*, ASME Publication 74-PVP-13, New York, 1974.
9. Knupp, P. & Steinberg, S., *Fundamentals of Grid Generation*, CRC Press, Boca Raton, FL, 1993.
10. Kohonen, T., *Self-Organization and Associative Memory*, 2d ed., Springer-Verlag, Berlin, 1988.
11. Thames, F. C., Thompson, J. F., Mastin, C. W. & Walker, R. L., Numerical solutions for viscous and potential flow about arbitrary two-dimensional bodies using body-fitted coordinate systems, *Journal of Computational Physics*, **24** (1977), 245–73.
12. Renka, R., Triangulation and bivariate interpolation for irregularly distributed data points, Ph.D. thesis, University of Texas at Austin, 1981.
13. Rhynsburger, D. Analytic delineation of thiessen polygons, *Geographic Analysis*, **5** (1973), 133–44.
14. Shephard, M. S. & Finnigan, P. M., Towards automatic model generation, in *State of the Art Surveys on Computational Mechanics*, A. K. Noor and T. J. Oden, eds., ASME, New York, 1989, pp. 335–366.
15. Thomasset, F., Appendix to Navier-Stokes problems, in *Navier-Stokes Problems*, R. Temam, ed., North Holland, Amsterdam, 1977.
16. Tubakman, T. & Exman, I., Towards real-time self-organizing maps with parallel and noisy inputs, in *Proceedings of the* 10*th Israeli Symposium on Artificial Intelligence*, *Computer Vision and Neural Networks*, Ramat Gan, Israel, 1993, 155–64.
17. Yousef, M., Automatic mesh generation using self-organizing neural networks, master's thesis, University of Haifa, 1996 (in Hebrew).