Available at
www.ComputerScienceWeb.com
POWERED BY SCIENCE @ DIRECT®

Applied Soft Computing 3 (2003) 37–51

Applied Soft Computing
www.elsevier.com/locate/asoc

Towards automating the finite element method: a test-bed for soft computing[☆]

Larry M. Manevitz[a,b,c,*], Dan Givoli[d]

[a] Department of Computer Science, University of Haifa, Haifa, Israel
[b] Department of Experimental Psychology and the Institute of Mathematics, Oxford University, Oxford, UK
[c] Department of Computer Science, Royal Holloway, University of London, Egham, UK
[d] Faculty of Aerospace Engineering, Technion—Israel Institute of Technology, Haifa, Israel

Received 15 March 2002; received in revised form 24 June 2002; accepted 24 July 2002

Abstract

A program to automate the finite element method (FEM) using various soft-computing techniques is presented. The overall program is discussed, and the implementations of three specific sub-problems (mesh placement, node numbering, and adaptive meshing) are described. It is also argued that the overall architecture of an "intelligent finite element package" can serve as a "test-bed" for many soft-computing techniques.

Keywords: Finite element method; Soft computing; Test-bed; Neural networks

1. Scientific background

The finite element method (FEM) is a computationally intensive method for the numerical solution of partial differential equations. It is a widely used tool and in many cases is the method of choice. This is especially (but not exclusively) true in structural engineering and solid continuum mechanics. The finite element procedure has been proved most effective when applied to linear boundary value problems in spatial domains which are complicated geometrically [15].

Basically (and oversimplifying), the FEM works by deciding a priori on a certain kind of simple approximation to the solution, and by dividing up the region of solution into small "elements", and allowing the parameters of the simple approximations to vary from element to element. The requirement that the individual local solutions remain consistent together with, e.g. boundary conditions, results in linear constraints on the parameters. These are then solved by standard linear algebra techniques.

However, in practice one can not use the FEM as a "black-box" solver; i.e. it is not sufficient to know the governing equations, the geometry, and the boundary and initial conditions in order to obtain high-quality numerical results. It is well appreciated among finite element users in industrial and scientific communities that the successful application of this technique requires substantial amount of experience and expertise in order to make the computation feasible and the results accurate at the same time. This is true with regard to any of the large commercial finite element packages

[☆] Partially supported by joint U. Haifa-Technion Research Grant. Gratitude is expressed to Profs. Zilber, Rolls and Gutin for their hospitality during a sabbatical of the first author.
[*] Corresponding author.
E-mail addresses: manevitz@cs.haifa.ac.il (L.M. Manevitz), givolid@aerodyne.technion.ac.il (D. Givoli).

1568-4946/03/$ – see front matter © 2003 Elsevier Science B.V. All rights reserved.
doi:10.1016/S1568-4946(03)00004-8

available currently, although various codes may have a different amount of flexibility.

This is because there is a very large number of parameters that need to be chosen; computational limitations make the choice of these parameters crucial for successful use. These are knowledge-based requirements; and current usage requires much human expertise. There are no known effective algorithms that can replace a human user in all cases; and many of the problems are known to be computationally intractable in the general case [9].

On the other hand, the complexity of some of the problems are beyond the realm of efficient manual control (e.g. in more than two dimensions or in quite complicated situations like parallel implementations which are not directly amenable to human intuitions).

In recent years, there has been a large development of tools for artificial intelligence, neural networks (NNs), fuzzy logic and related disciplines (called by some "soft computing" [34–36]), which our work and analysis indicate are applicable to the problems under discussion here. Each problem can in fact be attacked by a number of methods, but *usually*, a certain approach suggests itself as the most promising one in each case. Thus, expert system technology [14] seems most appropriate to replace human numberers, self-organizing neural networks [18] and fuzzy logic

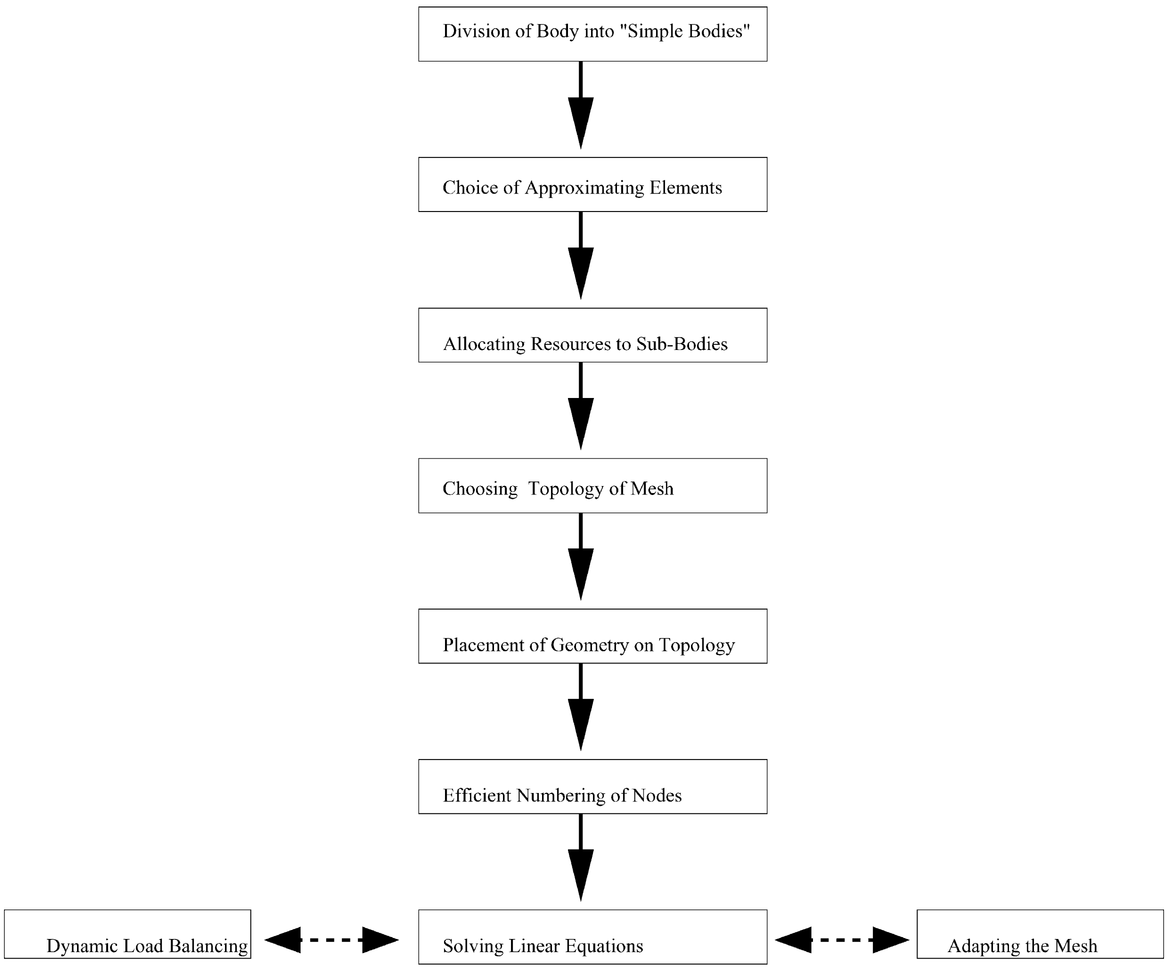

Division of Body into "Simple Bodies"
Choice of Approximating Elements
Allocating Resources to Sub-Bodies
Choosing Topology of Mesh
Placement of Geometry on Topology
Efficient Numbering of Nodes
Dynamic Load Balancing
Solving Linear Equations
Adapting the Mesh

Fig. 1. Some general tasks in the finite element method.

"critics" [34] seem appropriate for mesh placement, neural network non-linear predictors are appropriate for dynamic mesh placement for solutions to time-dependent PDEs [33], distributed artificial intelligence [28] and genetic algorithms [12] can be appropriate for load balancing in parallel computation, and so on. The specific problems and tools we have in mind are described in Fig. 2.

Our main goal in working on the finite element method is, of course, the automation of the method; and any efficient means towards that is welcome. In this paper, we report on our results so far.

In addition, we have come to realize [20,21] that the FEM serves as a rich test-bed appropriate for the serious use of these techniques in real world (i.e. non-"toy") problem settings. Evaluating these techniques in realistic settings is in itself an important research goal. In fact, our initial work has already resulted in advancing somewhat [22] one of the most classical neural network algorithms as a result of evaluating the needs in the realistic setting.

We point out that since one can compare results with analytic ones in certain settings, it allows for quantitative evaluation of the effectivity of the techniques. In principle, this allows one to compare and evaluate competing techniques and directions both between themselves and with current commercial packages.

By a test-bed we have in mind a system that allows one to implement a variety of techniques on various problems, run it easily on "real-world" applications, and be able to do some sort of evaluation/comparison studies with other techniques.

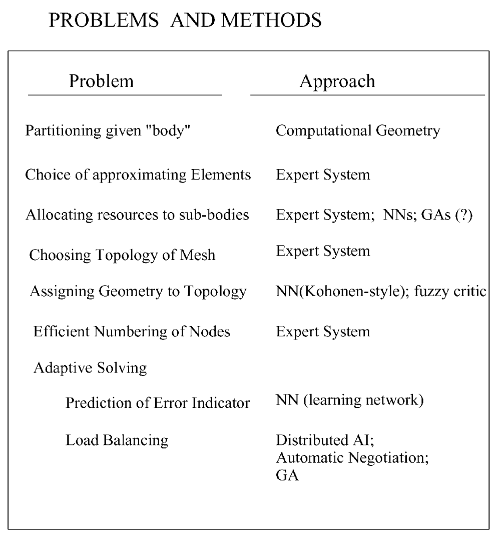
Intelligent automation of the FEM can be appropriate because of the following reasons:

- There are a wide variety of optimization/satisfaction sub-problems to be solved. Various techniques are applicable to the different problems.
- The architecture of the automation is modular. This means that each of the sub-problems can be solved more or less independently. ("More or less" means that when all the solution steps are in place, one has to consider another optimization problem concerning trade-offs. However, this affects only the potential optimality of the FEM solution, and not the effectivity of the individual steps.) Each of the steps can be evaluated individually as to how it affects the global quality of the numerical solutions.
- The nature of the FEM, a numerical solver for partial differential equations, makes it easy to generate test problems. In fact, any partial differential equation with boundary conditions defines a test case. This means it is almost as easy to work on real world problems as on artificial simple or "toy" ones. One can also provide test cases with analytic solutions to provide a natural "gold standard" for evaluations.
- The importance of the FEM means that there are commercial codes available for comparison.

1.1. Some sub-problems for the finite element method

In Fig. 1, we list some of the human-based tasks that must be performed in using the FEM. Fig. 2 lists some of the corresponding possible techniques.

So far, together with our students, Akram Bitar, Miha Margi and Malik Yousef, we have implemented three of these solutions: (temporal mesh adaptation using feed-forward neural networks, automated mesh numbering via an expert system and mesh placement via a self-organizing neural network). In the following sections, we sketch the results of these sub-problems (full details appear or will appear elsewhere [20,21,23]).


PROBLEMS AND METHODS

Problem	Approach
Partitioning given "body"	Computational Geometry
Choice of approximating Elements	Expert System
Allocating resources to sub-bodies	Expert System; NNs; GAs (?)
Choosing Topology of Mesh	Expert System
Assigning Geometry to Topology	NN(Kohonen-style); fuzzy critic
Efficient Numbering of Nodes	Expert System
Adaptive Solving	
Prediction of Error Indicator	NN (learning network)
Load Balancing	Distributed AI; Automatic Negotiation; GA

Fig. 2. Table of problems and soft-computing solution approaches.

2. Dynamic mesh adaptation for temporal PDEs

Time is often treated distinctly from spatial dimensions in the solution phase of PDEs. That is, the typical method of choice for solution of such equations is *not* to treat time as simply another dimension, but to "simulate time"; i.e. to repeatedly solve the equations for different times; using the previous solution as the starting conditions for the next one.

However, in a dynamic system, this implies that one should not use the same mesh at different times. For example, when the solution of a hyperbolic problem involves a shock wave that propagates through the mesh, the location of the critical regions, namely the shock vicinity, keeps changing in time. To get optimal numerical results with a fixed computational effort, it is important that there be more mesh elements near this shock vicinity at each time step. Another example is a problem of fluid flow in a cavity, where flow cells are generated and undergo continuous changes in their shapes and size as time proceeds [11].

Thus, the mesh choice should be dynamic; varying with time. In current usage, the method is to use the gradient of the solution at time t_n to indicate where the mesh should be modified (where it should be refined and where it can be made coarser) at time t_{n+1}. The work [11] used this mechanism.

However, this suffers from the defect that one is always one step behind. If the area of interest is propagating (a common phenomenon) then one may be always refining directly behind the most interesting phenomenon. This is also assuming that one does not miss the "action" altogether. One can look at this as a special instance of a control problem.

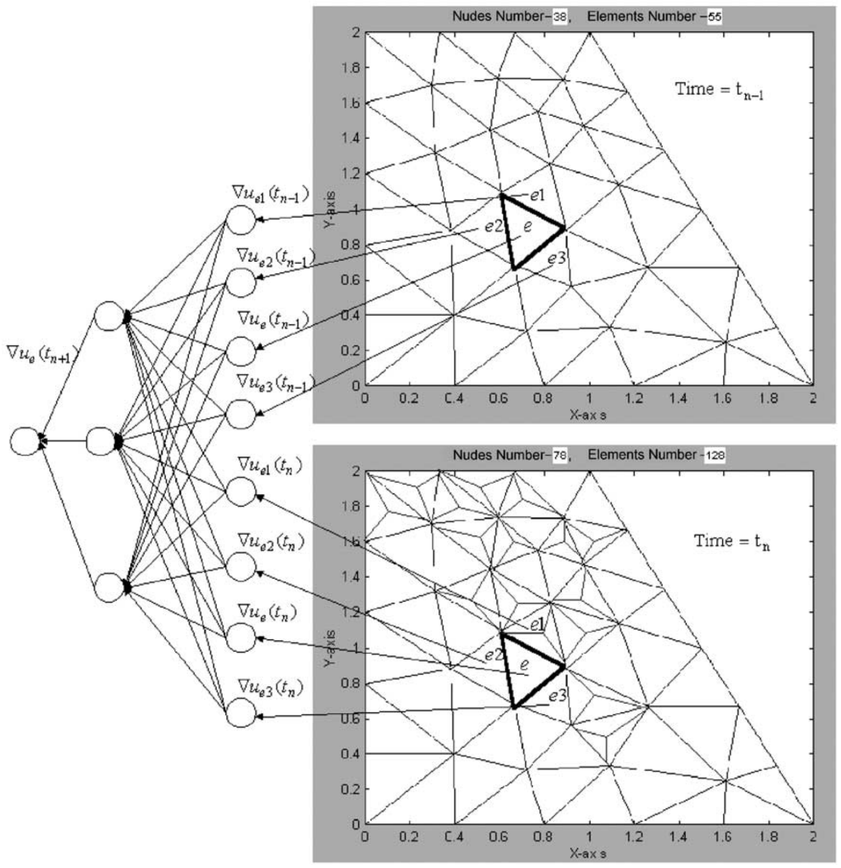


Fig. 3. Using neural networks to forecast future FEM gradient values. Gradient values for two previous times and all the neighboring elements are used as input to the network.

In this section, we present a new method based on artificial neural networks that enables one to predict the critical regions ahead of time, thus allowing the adaptation of the mesh at the appropriate times and places. Our mechanism relies on the universal approximator [5,8,19,24] property of feed-forward neural networks. In the current version, we train the network to serve as a function generating the next gradient value at an element in the mesh, based on the current and previous gradient values of the element and its immediate neighbors. This is a proper time series problem and the time series neural networks can be used to predict the future gradient values. Fig. 3 illustrates this concept. (The idea of using a neural network as a "plant" predictor has been developed in the context of control theory and signal analysis [32,33].)

We used two different networks, one for boundary elements and one for interior elements. Using a small number of networks means that a very few time steps generate much data for training. (That is, many different elements give training data to the same network.)

Thus, there are two steps to our methodology: (1) training the neural network to predict the indicators (at least) one step in advance; and (2) applying the indicators to the refinement in the FEM solution.

In our experiments, we examined the numerical results of applying this procedure using the FEM on different time-dependent PDE problems using different parameters for the NN algorithm and comparing this with: (i) FEM with no adaptation; and (ii) FEM using the "standard" adaptation via the current gradient indicator.

Table 1
One-dimensional examples

Method	No. of refined elements	L^2 norm		L^∞ norm	
		Maximum	Average	Maximum	Average
NN modifier[a]	70	0.15756	0.0869	0.1653	0.1056
Standard modifier	70	0.1826	0.1022	0.1914	0.1222
No adaptation	0	2.5332	1.0053	3.4708	1.0643
NN modifier[b]	98	0.4423	0.1928	0.5190	0.2342
Standard modifier	91	0.6671	0.2686	0.8230	0.3142
No adaptation	0	1.4622	0.6985	1.6288	0.6456

Comparison between FEMs run with: (i) the neural network predictor of the gradient measure; (ii) "standard" refinements using the gradient measure; and (iii) no adaptation.

[a] Example 1:

$$\frac{\partial^2 u}{\partial t^2} = \frac{\partial^2 u}{\partial x^2}, \quad 0 \le x \le 10,$$

$$u(0,t) = 0 \quad \text{and} \quad u(10,t) = 0.$$

$$u(x,0) = \begin{cases} 1 - |1 - x|, & 1 \le x \le 2, \\ 0, & \text{otherwise}. \end{cases}$$

Number of initial elements: 10; time: 12; time step: 0.08; threshold for refinement = 0.08 (gradient); improvement: L^2 error norm = 15%; L^∞ error norm = 13.6%.

[b] Example 2:

$$\frac{\partial^2 u}{\partial t^2} = \frac{\partial^2 u}{\partial x^2}, \quad 0 \le x \le 25,$$

$$u(0,t) = 0 \quad \text{and} \quad u(25,t) = 0,$$

$$u(x,0) = \begin{cases} \exp(\frac{1}{2}(-(x-5)^2)), & 0 \le x \le 10, \\ 0, & \text{otherwise}. \end{cases}$$

Number of initial elements: 15; time: 25; time step: 0.12; threshold for refinement = 0.2 (gradient); improvement: L^2 error norm = 28%; L^∞ error norm = 25%.

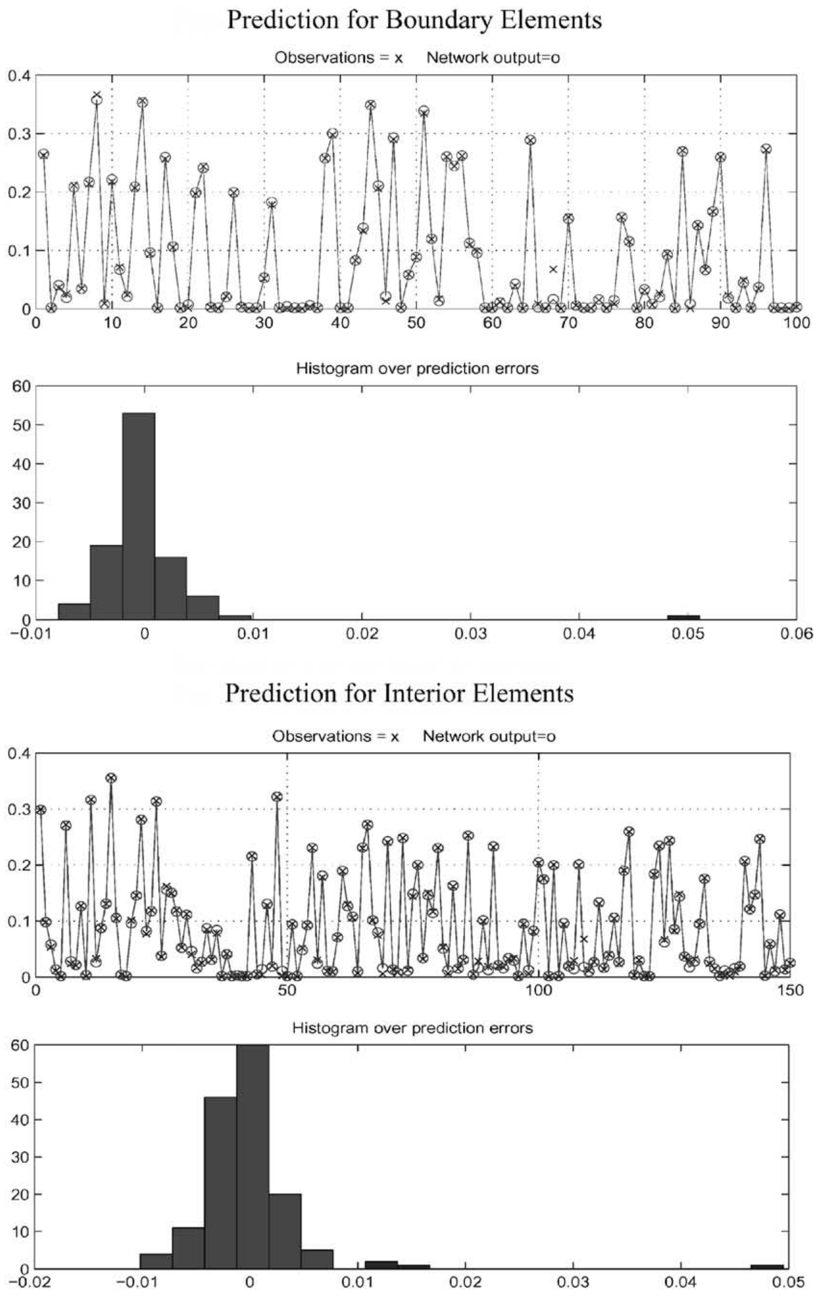


Fig. 4. Time series prediction test for the one-dimensional wave equation (see text). Blue (×) indicates test values; red (○) the network response.

In this study, the MATLAB's Neural Network Toolbox [6] was used for designing and training the neural network. In the examples tested, the results are fairly dramatic. First, using the Levenberg–Marquardt training algorithm [27] (a variant of Gauss–Newton) the training was both quite swift and exceptionally accurate (Fig. 4). Second, the improvement in the FEM numerical results (as compared with the "standard" gradient adaptive method) reached as high as 28% on some examples; and never fell significantly below the standard method. (The variance in the improvement depends on the shape of the wave; and is to be expected. That is, for some waves it is more important to predict the gradient than others.)

We used two different networks, one for boundary elements and one for interior elements. The architecture of both networks was six input units (corresponding to the value of the gradient of the element and its two neighbors in the current and previous times), six hidden units (with tan-sigmoid transfer function), and one output unit (with linear transfer function) that gave the prediction of the output value (Fig. 3).

Complete details on this work will appear in [23]. Here we give some sample examples. Fig. 4 shows the results of a typical prediction test for interior and boundary elements. Training took about 117 epochs to converge to extremely small error (about 0.00024) in the interior elements prediction. Results for the boundary elements were similar. When applying this modifier to the FEM mesh, the numerical improvement over the "standard" gradient modifier varied from no significant improvement to an improvement of more than 25% (both in the L^2 norm and in the L^∞ norm).

Table 1 presents the comparison results for two sample examples, where the initial condition of the wave is a Gaussian. The analytical solution is well known for these types of problems and it depends on the initial and boundary conditions. The wave splits into two waves (with the same width but half the height) that travels to the left and to the right with speed $c = 1$. When such a travelling triangle reaches the edge, it turns over and returns upside down (Fig. 5). The NN-modified solution and the "standard" gradient modifier are displayed in Fig. 6. Compare the graphs in Fig. 6. Observing the areas indicated in the figure, one can see that the NN chose to place its resources in the correct places. Looking at the refinement markings (in red or dots on the x-axis); one can see that, as suggested by our theory, the NN is keeping pace with the development of the solution, whereas the "standard" method is always on step behind, which at critical locations causes increased numerical error.

Since such examples have analytic solutions, we were able to calculate the actual numerical errors of each of the methods.

Two sample examples done in two dimensions are presented in Table 2. Note that: (i) the prediction of the gradient was very accurate (Figs. 7 and 8); and (ii) the improvement in the FEM numerical results were around 6.5% over the standard gradient methods.

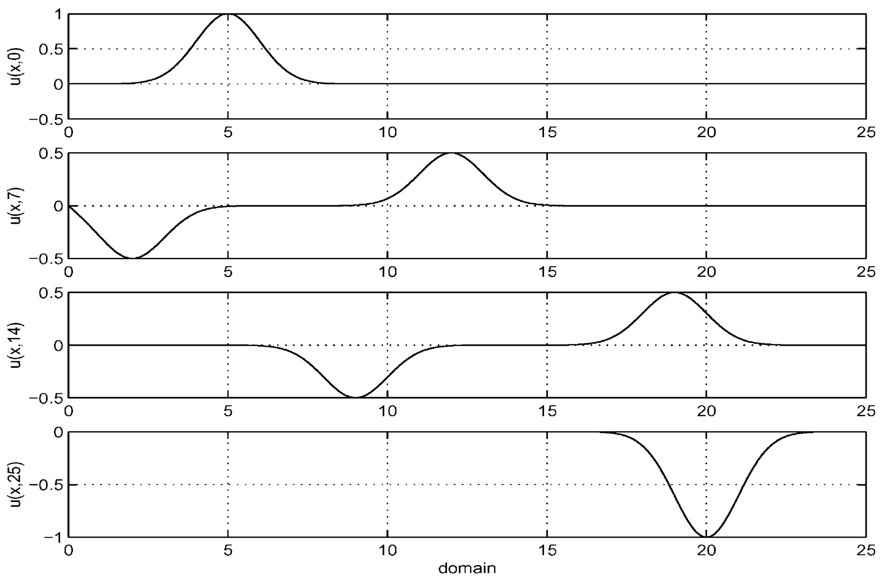


Fig. 5. Analytic solution.

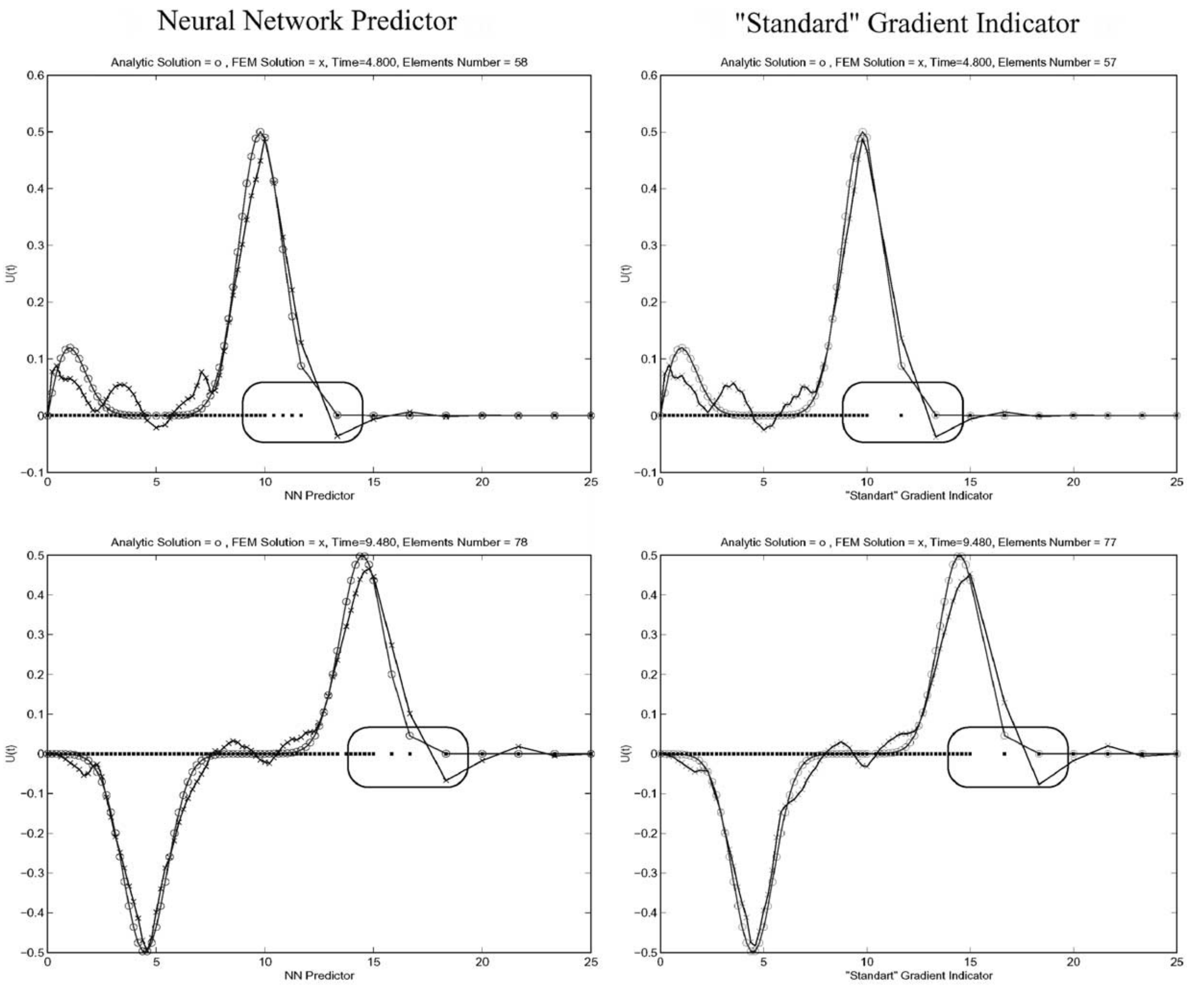


Fig. 6. Results from FEM on one-dimensional wave equation. The left figures are refined with the NN predictor. Also indicated is the analytic solution. The right figures are refined with the “standard” gradient indicator. Compare the segments of the curves on the left (enclosed rounded rectangles) with the corresponding ones on the right to see how the NN predictory focuses the resources in the correct places.

In summary, a version of a NN modifier for the FEM mesh has been implemented, which is designed to adaptively change the mesh based on a *prediction* of the gradient. In experimental work, we have shown that the NN can accurately predict the gradient and applying this mesh results in a substantial numerical improvement.

3. Heuristically numbering the nodes

When applied to linear boundary value problems, the finite element discretization of the governing partial differential equations leads finally to a linear system of algebraic equations [15],

$$\boldsymbol{K}\boldsymbol{d} = \boldsymbol{F}$$

where $\boldsymbol{K}$ is the so-called *stiffness matrix*, $\boldsymbol{F}$ the load vector, and $\boldsymbol{d}$ the vector of unknowns. The dimension of the system (1) in typical industrial applications is of order 100 or 1000 or even 10,000. This dimension arises from the chosen discretization of the spatial domain; i.e. the spatial domain is divided into *elements* connected to each other by nodes. These nodes are numbered and $\boldsymbol{K}_{ij} \neq 0$ only if nodes i and j belong

Table 2
Two-dimensional examples

Method	No. of refined elements	Average L^2 norm	Average L^∞ norm
NN modifier[a]	803	0.4057	0.4846
Standard modifier	803	0.4314	0.5029
NN modifier[b]	246	0.2962	0.3359
Standard modifier	232	0.3256	0.3807

Comparison between FEMs run with: (i) the neural network predictor of the gradient measure; and (ii) "standard" refinements using the gradient measure.

[a] Example 1:

$$\frac{\partial^2 u}{\partial t^2}=\frac{\partial^2 u}{\partial x^2}+\frac{\partial^2 u}{\partial y^2},\quad -1\le x\le 1;\quad -1\le y\le 1,$$

$$u(-1,y,t)=0 \quad\text{and}\quad u(1,y,t)=0,\quad \text{for}\quad -1\le y\le 1,$$

$$u(x,-1,t)=0 \quad\text{and}\quad u(x,-1,t)=0,\quad \text{for}\quad -1\le x\le 1,$$

$$u(x,y,0)=\begin{cases}15x(x+1)y(y+1), & -1\le x\le 0;\quad -1\le y\le 0,\\ 0, & \text{otherwise.}\end{cases}$$

Number of initial elements: 28; time: 3; time step: 0.05; improvement: L^2 error norm $=6\%$; L^∞ error norm $=3.6\%$; threshold for refinement $=1$ (gradient).

[b] Example 2:

$$\frac{\partial^2 u}{\partial t^2}=\frac{\partial^2 u}{\partial x^2}+\frac{\partial^2 u}{\partial y^2},\quad -1\le x\le 1;\quad -1\le y\le 1,$$

$$u(-1,y,t)=0 \quad\text{and}\quad u(1,y,t)=0,\quad \text{for}\quad -1\le y\le 1,$$

$$u(x,-1,t)=0 \quad\text{and}\quad u(x,-1,t)=0,\quad \text{for}\quad -1\le x\le 1,$$

$$u(x,y,0)=\arctan\left(\cos\left(\frac{\pi}{2\pi}\right)\right),$$

$$\frac{\partial^2 u}{\partial t^2}(x,y,0)=3\sin(\pi x)\exp\left(\sin\left(\frac{\pi}{2\pi}\right)\right).$$

Number of initial elements: 28; time: 3; time step: 0.08; threshold for refinement $=2.2$ (gradient); improvement: L^2 error norm $=9\%$; L^∞ error norm $=11\%$.

to a common element.Since the mesh is typically imposed on a physical domain and thus can be thought of as a planar or spatial graph, most $\boldsymbol{K}_{ij}$ are zero, i.e. $\boldsymbol{K}$ is a *sparse* matrix. This, in principle, reduces the computational resources needed to solve such a system; however, this is sensitive to how "near-diagonal" all the non-zero entries are.

The discussion above implies that the order in which the nodes are numbered is crucial, since this numbering determines how near-diagonal $\boldsymbol{K}$ is. Note first that optimal numbering of a given mesh is an NP complete problem [9]. Thus, for a mesh containing a large number of nodes and elements, it is of course impractical to find an exact solution for the optimization problem by performing a full enumeration. However, there are so-called "algorithmic" solutions that have been incorporated in commercial software. Anecdotal evidence about this software reports that: (i) a human expert can often outperform it; (ii) at best, such algorithms typically result in rather complicated numberings involving substantial "windings" and for some applications there is an advantage in the simplicity of the numbering. (These algorithms include the methods described by Cuthill and McKee [4], King [17], Collins [3], Akin and Pardue [1], Gibbs [10], Razzaque [26], Pina [25] Sloan and Randolph [30], Fenves and Law [7], and Sloan [31].)

Accordingly, we decided to test the feasibility of developing an expert system to try and mimic the heuristic mechanisms of a proficient human numberer.

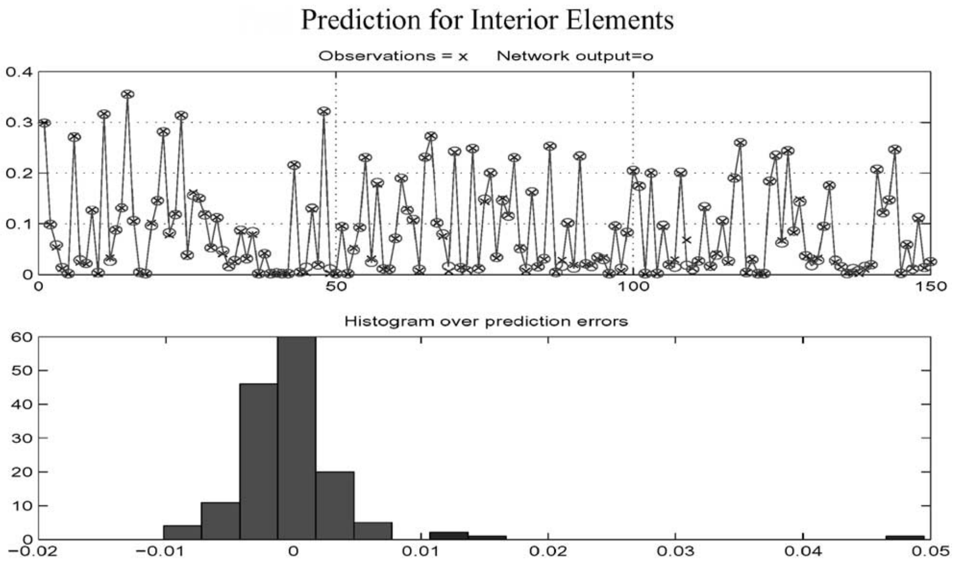


Fig. 7. Time series prediction test for the two-dimensional wave equation. Blue (×) indicates test values; red (○) the network response.

Actually, only some of the possible heuristics were implemented, but the system proved to compare favorably with the human user on a variety of test examples. (Full details appear in [20].)

In order to state the optimization problem mathematically, we introduce some notation. Let the dimension of the matrix $\boldsymbol{K}$ be $N \times N$. Let b_i be defined as $b_i = i + \max_{K_{ji} \neq 0} j$. In other words, b_i is the height of column i starting from the diagonal and up to the skyline. Now, let the *average half bandwidth* (AHB) and the *root mean square bandwidth* (RMSB) be defined as

$$\mathrm{AHB} = \frac{1}{N}\sum_{i=1}^{N} b_i$$

and

$$\mathrm{RMSB} = \left(\frac{1}{N}\sum_{i=1}^{N} b_i^2\right)^{1/2}$$

It is apparent that the bandwidth depends on the nodal numbering system. This dependence becomes strong for large meshes. The optimization problem under consideration can thus be stated as follows: Find a numbering system for the nodes (from 1 to N) such that AHB (or RMSB) will be *minimal*.

3.1. A brief description of the expert system characteristics

One of the hardest steps in producing an expert system is to construct the heuristics according to which the expert system will perform, and which will mimic the considerations of a human expert. To this end, we note the following functions typically performed when numbering the nodes of a finite element mesh manually and briefly comment on our system's approach to these tasks.

1. Preparation of numbering strategies for "paradigm simple blocks" with different geometries and topologies. Varying parameters of freedom are associated with each strategy. These strategies are achieved by experience, by trial and error, and sometimes by full analysis. This step is not "performed" for each mesh separately, but is rather a database of knowledge an expert has accumulated. Examples of simple blocks are rectangles, annuli, discs.
2. Subdivision of the mesh into a disjoint union of simple blocks. Our system does not perform this, but receives it as given.
3. Choosing the order in which these blocks are picked for numbering.

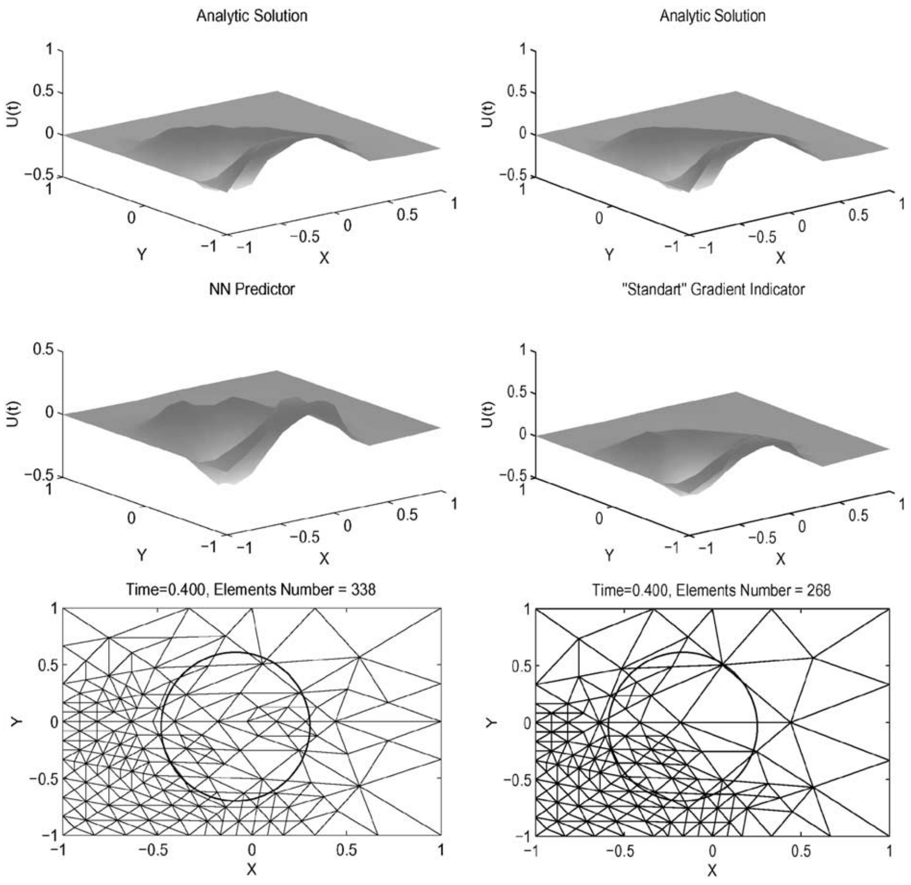


Fig. 8. FEM result. Two-dimensional wave equation example 1 (Table 2). The left figures are refined with the NN predictor. The right figures are refined with the "standard" gradient indicator.

In this step, the order of the simple blocks is determined. To optimize the numbering a user tries to keep the node numbers as continuous as possible. However when passing from block to block, this is often impossible. Thus, the goal is to keep the "jumps" across block interfaces as small as possible. In the discussion that follows, a *component* is a topological component, i.e. a maximal connected sub-body.

In the implemented system, the procedure for block ordering is as follows:

- Choose the first block to be numbered as the largest simple block, based on the number of nodes.
- Remove this simple block from the mesh. (This may cause the remaining mesh to have several disconnected components.)
- Place all the resulting separate components in increasing order in a stack.
- Until the stack is empty do the following:
    - ○ Remove the smallest component from the stack.
    - ○ From the chosen component, choose the next simple block to be numbered in the following way:
        - – Consider all simple blocks in the component with an interface with the previously numbered block (there are always such).

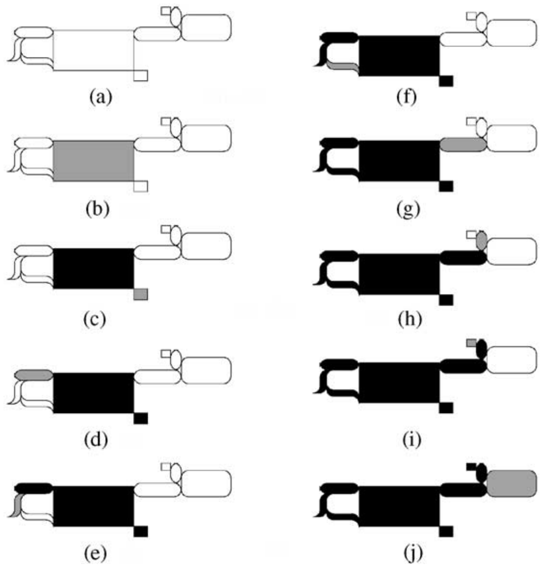


Fig. 9. Ordering the simple blocks.

– Choose the one with the largest interface.
– Remove the chosen block from the component. (This may divide the remaining blocks into several connected components.)
– Place the new components on the stack in increasing order.

See Fig. 9 for an example of how the blocks are numbered.

4. For each block, choosing a numbering strategy, depending on the topology and the geometry of the block. That is, choosing the best match to a paradigm simple block. Essentially, this has two parts: (1) solution of a pattern recognition problem; i.e. given a simple block find which of the paradigm simple blocks it is closest to; and (2) after identification, choosing a numbering method for the block form amongst the possible methods mentioned in step 1.

   In the implemented version, all the blocks are assumed to be pseudo-rectangles and there is only one numbering strategy for numbering rectangles, so this step is to a large extent vacuous.

   Notwithstanding, it remains for the system to perform the following analysis: (a) to decide on the orientation of the block, regarded as having a rectangular shape; and (b) to be able to ignore small perturbations. The first is done using an algorithm *involving* the convex hull of the block and regression; the second is accomplished using a heuristic (also *involving* the convex hull) that defines a “grain size”.
5. For each block, determining the free parameters associated with the strategy chosen for that block, such as the node from which to start the numbering. For the rectangular strategy handled in the implemented version, the only free parameter is the node from which to start the numbering, and there are four possibilities, corresponding to the four corners of the pseudo-rectangle. This is selected by a heuristic that is based on both topological and geometrical considerations.

It is interesting and crucial to note that while the mesh is a purely topological object as far as node numbering is concerned, the user decides on the division based not only on topological criteria, but also on *geometric* criteria. This is an obvious logical fallacy and it is easy to devise artificial mesh examples where the results are extremely bad. Nonetheless, for real meshes, the results seem acceptable.The reason why human users can be so successful in numbering meshes although basing their decisions on geometric considerations is that they intuitively rely on the fact that *actual meshes are designed with implicit geometric constraints*. For example, a good practice in mesh design is to use elements which have aspect ratios close to unity, namely quadrilaterals and triangles which are nearly equilateral. Another good practice is to pass from a crude region (large elements) to a refined region (small elements) in a gradual manner [16].

Our expert system tries to mimic a human user; therefore, it uses geometric considerations as well as topological ones. This is in contrast to the “algorithmic approach”, where only the topological properties of the mesh play a part in the node reordering procedure.

The system was tested on a number of meshes taken from the literature; here, we mention the largest mesh tested to date similar to one that appears in [37], which had 359 nodes and 559 triangular elements (Fig. 10).

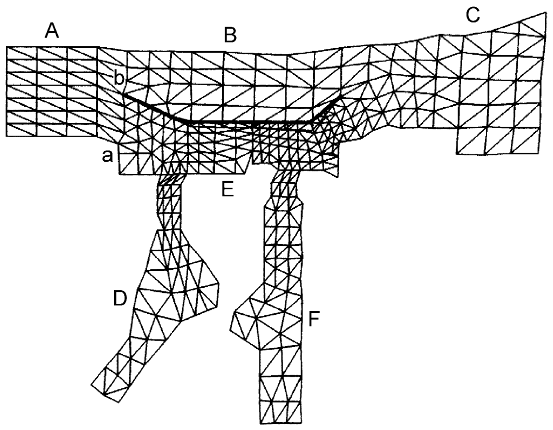


Fig. 10. An example with 359 nodes and 559 elements. Half bandwidth for a random numbering was 210.2, half bandwidth for a human expert numberer was 15.9, half bandwidth for the heuristic numberer was 17.6.

Here, the average half bandwidth of the system was 17.6; that of the human expert was 15.9 while for comparison a random numbering resulted in a value of 210.2.

Some other meshes were taken from applications in large deformation continuum mechanics discussed in [13] and solved by the finite element code NIKE2D. The results were are quite comparable to that of the human expert. (In one case, the expert system actually outperformed the human expert.)

The results show that although the human expert performs better than the expert system, the differences between the two AHBs are reasonably small. Other examples have been tested as well and the results compared favorably with those of a human expert.

4. Mesh placement via self-organizing neural networks

Once a topologic mesh has been defined (a problem we have suggested is appropriate for an expert system), the mesh has to be given its geometry; i.e. it has to be placed appropriately on the body. Moreover, there are points and regions which one wants to cover with a finer mesh than other areas. These are regions or points of “interest” where the approximation used in the finite element method is intrinsically worse. Essentially, it is an optimization problem; given a fixed amount of computation that one wants to expend, which translates into a fixed amount of elements; how should one best distribute these elements so as to obtain the best approximation using the finite element method. A better mesh results in a better approximation.

There are several requirements for the quality of a mesh. For example, one wants the proportions of the elements to be as close as possible to those with good aspect ratios (the aspect ratio is the ratio of the radii of circumscribed circle to that of the inscribed circle: “good” means here close to one). For a quadrilateral then, for the best results from the finite element method one wants quadrilaterals close to squares and triangles close to equilaterals. In addition, one wants the change in size between elements determined by the mesh to be gradual. Quantitatively, one wants to keep the ratio of radii of circumscribed circles of adjacent elements to be close to 1; globally, it means that the maximum and minimum of such ratios of all pairs of adjacent elements should be as close to 1 as possible.

Moreover, in a typical finite element mesh, the density of the nodes is taken to vary from being very high near certain critical regions or points to more sparse where simpler approximations will suffice.

This can be handled using a modification of Kohonen’s self-organizing the neural network approach [21,29] by setting the probability density function of the input to correspond to the desired density of the network. The “self-organizing” feature of the Kohonen map will then place the highest density of nodes according to the sampling of data according to this density function. In this implementation, one identifies the weight of the neurons with the geometric coordinates of the body. The Kohonen map results in an equiprobably response map of the neural net (constrained by its given topology) subject to the sample data.However, in addition one requires that a mesh be placed so that boundary points of the mesh fall on the boundary of the domain. Unfortunately, since the boundary has measure zero in the domain, the Kohonen algorithm will not meet this requirement. (The topological preserving properties of the Kohonen map will force boundary points of the neural network to remain on the boundary of the *network*.) We were able to solve this problem by defining two Kohonen maps; one between the one-dimensional boundaries; and one between the full two-dimensional spaces and

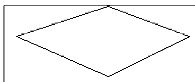
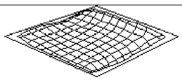
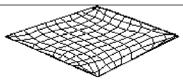
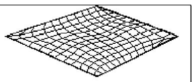
			
A. Domain of Application	B. Result of Standard Kohonen Mapping	C. Result of Sampled Boundary Algorithm	D. Result of Interwoven 1-D and 2-D Kohonen Map

Fig. 11. Comparison of different algorithms.

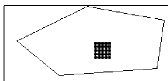
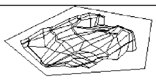
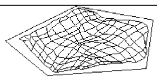
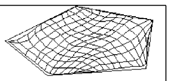
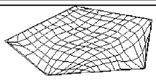
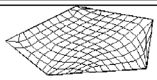
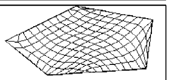
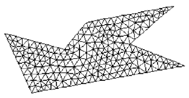
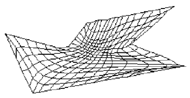
			
Iteration 0; Initial Setup	Iteration 500; Quality = 288.10	Iteration 2000; Quality = 237.78	Iteration 4500; Quality = 226.00
			
	Iteration 6000 Quality = 222.81	Iteration 12000 Quality = 207.79	Iteration 30000 Quality = 202.46

Fig. 12. A sequence of snapshots of a mesh being placed via a NN algorithm.

doing an appropriate interweaving of the two. See Fig. 11 for a comparison. (Diagram B is the basic Kohonen map; diagram D is our solution; diagram C is result obtained by a more direct oversampling method we tried.) This is discussed further in [22]. Fig. 12 gives a view of the algorithm in action.

In a series of experiments, this algorithm was compared with one of the most popular finite element packages over a series of problems, and was found to be generally superior. Fig. 13 gives a sample comparison with the PLTMG [2] package. Full details of this experiment can be found in [21].

	
PLTMG 249 nodes	NN 225 nodes; Quality = 278.713715; ''Hot-spot" (2,2) near center; 46650 Iterations

7-Sided Non-Convex Domain PLTMG (249 nodes, 437 elements) N N (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. 13. A typical comparison between the NN and a commercial package.

References

[1] J.E. Akin, R.M. Pardue, Element resequencing for frontal solutions, in: J.R. Whiteman (Ed.), Mathematics of Finite Elements and Applications, Academic Press, New York, 1975.

[2] R.E. Bank, PLTMG: A Software Package for Solving Elliptic Partial Differential Equations, SIAM Publications, Philadelphia, 1994.

[3] R.J. Collins, Bandwidth reduction by automatic renumbering, Int. J. Numer. Methods Eng. 6 (1973) 345–356.

[4] E. Cuthill, J. McKee, Reducing the bandwidth of sparse symmetric matrices, in: Proceedings of the ACM National Conference, ACM, 1969.

[5] G. Cybenko, Approximation by superpositions of a sigmoidal function, Math. Control Signals Syst. 2 (1989) 303–314.

[6] H. Demuth, M. Beale, Neural Network Toolbox User's Guide, The Math Works Inc., Natick, 2000.

[7] S.J. Fenves, K.H. Law, A two-step approach to finite element ordering, Int. J. Numer. Methods Eng. 19 (1983) 891–911.

[8] A.R. Gallant, H. White, On learning the derivatives on an unknown mapping with multilayer feedforward networks, Neural Networks 5 (1992) 129–138.

[9] M.R. Garey, D.S. Johnson, Computers and Intractability: A Guide to the Theory of np-Completeness, Freeman, New York, 1979.

[10] N.E. Gibbs, A hybrid profile reduction algorithm, ACM Trans. Math. Software 2 (1976) 378–387.

[11] D. Givoli J.E. Flaherty, M.S. Shephard, Simulation of Czochralski melt flows using parallel adaptive finite element procedures, Modell. Simul. Mater. Sci. Eng., in press.

[12] D. Goldberg, Genetic Algorithms, Addison-Wesley, New York, 1989.

[13] J.O. Hallquist, NIKE2D—a vectorized, implicit, finite deformation, finite element code for analyzing the static and dynamic response of 2-d solids, Tech. Report UCID-19677, Rev. 1, University of California, 1967.

[14] F. Hayes-Roth, D.A. Waterman, D. Lenat (Eds.), Building Expert Systems, Addison-Wesley, New York, 1983.

[15] T.J.R. Hughes, The Finite Element Method, Prentice-Hall, New York, 1987.

[16] B. Irons, S. Ahmad, Techniques of Finite Elements, Ellis Horwood, Chichester, UK, 1980.

[17] I.P. King, An automatic reordering scheme for simultaneous equations derived from network systems, Int. J. Numer. Methods Eng. 2 (1970) 523–533.

[18] T. Kohonen, Self-Organization and Associative Memory, second ed., Springer-Verlag, Berlin, 1988.

[19] M. Leshno, V. Lin, A. Pinkus, S. Schocken, Multilayer feedforward networks with non-polynomial activation functions can approximate any continuous function, Neural Networks 6 (1993) 861–867.

[20] L. Manevitz, D. Givoli, M. Margi, Heuristic finite element node numbering, Comput. Syst. Eng. 4 (1993) 159–168.

[21] L. Manevitz, M. Yousef, D. Givoli, Finite element mesh generation using self-organizing neural networks, Microcomput. Civil Eng. 12 (1997) 4.

[22] L.M. Manevitz, Interweaving Kohonen maps of different dimensions to handle measure zero constraints on topological mappings, Neuroprocess. Lett. 5 (1997) 155–161.

[23] L.M. Manevitz, A. Bitar, D. Givoli, FEM mesh adaptation via time series prediction using NNS, in: Proceedings of WSC6, 2001, Springer-Verlag, 2002.

[24] M. Meltser, M. Shoham, L. Manevitz, Approximating functions by neural networks: a constructive solution in the uniform norm, Neural Networks 9 (6) (1996) 965–978.

[25] H.L. Pina, An algorithm for frontwidth reduction, Int. J. Numer. Methods Eng. 17 (1981) 1539–1546.

[26] A. Razzaque, Automatic reduction of frontwidth for finite element analysis, Int. J. Numer. Methods Eng. 15 (1980) 1315–1324.

[27] B.D. Ripley, Pattern Recognition and Neural Networks, Cambridge University Press, Cambridge, 1996.

[28] J. Rosenschein, Rules of Encounter, MIT Press, New York, 1994.

[29] O. Sarzeaud, Y. Stephan, C. Touzet, Finite element meshing using Kohonen's self-organizing maps, in: T. Kohonen, et al. (Ed.), Artificial Neural Networks, North-Holland, Amsterdam, 1991, pp. 1313–1317.

[30] S.W. Sloan, Randolph, Automatic element reordering for finite element analysis with frontal schemes, Int. J. Numer. Methods Eng. 19 (1983) 1153–1181.

[31] S.W. Sloan, An algorithm for profile and wavefront reduction of sparse matrices, Int. J. Numer. Methods Eng. 23 (1986) 239–251.

[32] B. Widrow, M. Lehr, 30 years of adaptive neural networks: perceptron, madaline, and backpropagation, Proc. IEEE 78 (9) (1990) 1415–1442.

[33] B. Widrow, S. Stearns, Adaptive Signal Processing, Prentice-Hall, New York, 1985.

[34] R.R. Yager, Essentials of Fuzzy Modeling and Control, Wiley, New York, 1994.

[35] R.R. Yager, B. Bouchon-Meunier, Fuzzy Logic and Soft Computing, World Scientific, Singapore, 1995.

[36] R.R. Yager, L.A. Zadeh (Eds.), Fuzzy Sets, Neural Networks, and Soft Computing, World Scientific, Singapore, 1995.

[37] O.C. Zienkiewicz, The Finite Element Method, third ed., McGraw-Hill, New York, 1977.