On Automatic Mesh Generation using Kohonen Maps

D. A. Lowther
CADLab, Electrical Engineering Department
McGill University, Montreal, Canada

W. Mai
Institut für Elektrische Maschinen, RWTH Aachen, Germany

***Abstract*—This paper presents the use of Kohonen self-organizing maps for automatic mesh generation. Previous algorithms are improved by adding a new search algorithm and boundary treatment. The results of small geometrical examples are shown. Investigations into parameter variations support the discussion about advantages and disadvantages of this method.**

***Index terms*—Self-organizing feature maps, Finite element methods, Neural networks, Automatic meshing, Computer aided engineering**

I. Introduction

The generation or modification of finite-element meshes is part of the analysis of electromagnetic devices. Meshes are produced from scratch or from existing nets, as in adaption processes. Both the algorithm for meshing and the handling of the information about the desired element sizes in the mesh region varies among different methods. There are mainly three different methods for the user to define the desired mesh density:

1. The user chooses a global maximum element size in the mesh region. This region is the entire geometry or parts of it defined by shape primitives such as polygons or circles.

2. The user sets the number of nodes on the outlines of the mesh regions. Although mesh generators take care of the inner parts, the user can not influence the meshing there. Especially large sections with varying desired element density have to be cut into smaller ones.

3. The user specifies a density of points, which are spread over the entire geometry, defines the desired element sizes, and indicates where regions of high density should result in a fine mesh. This kind of information can also be provided by automatic algorithms such as neural networks [1] or error calculations in adaptive meshing.

This paper examines the self organizing maps proposed by Kohonen [2], which use the third approach to representing the required density information, and are described in the next section.

Manuscript received November 3, 1997.
D. A. Lowther, e-mail D_Lowther@compuserve.com;
W. Mai, e-mail mai@rwth-aachen.de.

II. Method Of Self-Organizing Maps

A self-organizing map is a special neural network with partially connected neurons. According to a radius R a neuron has a defined number of neighboring neurons in what is called a "neighborhood" shown in fig. 1.

This paper interprets this interconnection topology as finite element information, i.e. the neurons are the nodes and the interconnections between them are the edges of triangular elements. Kohonen self-organizing maps have the property to preserve this during the learning process.

In a Kohonen network, two spaces are considered: the topological space of uniformly positioned neurons with fixed interconnections and the geometric space of physical locations of the nodes. The geometric node locations are represented by the weights applied to each neuron and the Kohonen network provides, in effect, a variable mapping between the two spaces over the entire mesh.

Thus, the weight vector for one neuron consists of two or three components for two-dimensional or three-dimensional meshes respectively [2]. These weights are the output of the network and are affected by the input to the network (the set of training points). Each training point provides a set of coordinates (weights) which are created based on the desired density distribution.

A. The learning algorithm

The neural network starts with an initial mesh obtained from a simple mapping algorithm, an adaptive meshing process or a neural network mesh generator [3]. A training set is given representing the desired mesh density shown in fig. 2. Then one training weight is presented to the network at a time. The training weight (i.e. coordinate

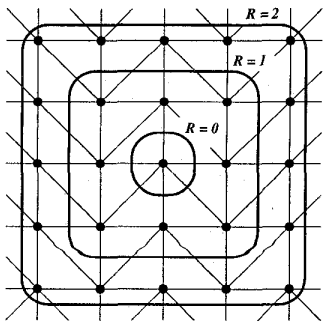

Fig. 1. The topological neighborhood (R=0, 1, 2)

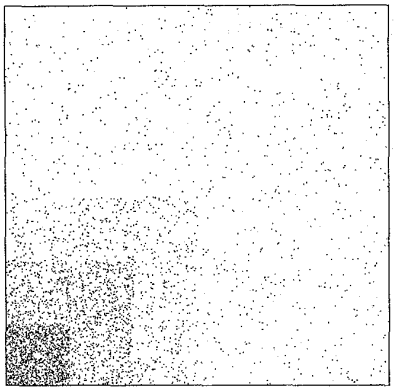

Fig. 2. Example of the density distribution

vector) is connected to the input of each neuron in the net simultaneously, which is different to the more classical feed forward net, where the training data are introduced to an input layer of neurons.

Unlike other neural networks the weights of the neurons are not calculated by the weighted sum of the input. Neither is every neuron updated when a training set is given [4]. The network updates only a "winning" neuron and a defined neighborhood.

The winning neuron N_c is defined as the closest neuron to the training point based on the Euclidean distance. This neuron is surrounded by a topological neighborhood shown for different radii in fig. 1. Both the winner and the members of the neighborhood of the radius R are updated according to the following rule:

$$w_{ik} = w_{ik}(old) + \alpha(e) \cdot (x_{tk} - w_{ik}(old)) \quad (1)$$

Where x_{tk} is the coordinate vector of the training point t, w_{ik} the weights of the considered neuron (in effect the current position in geometric space of the neuron) and $\alpha(e)$ a deceleration factor, which is gradually reduced as the training proceeds. k is the component of the coordinate vector.

The entire algorithm is described as follows: One training point is given to the net and the closest neuron has to be searched and updated along with its neighborhood. An epoch is defined as the period over which all the training points are presented to the net. The entire process tries to reduce the sum of minimum Euclidean distances e by repeating epochs until a stopping criterion is reached:

$$e = \sum_t \sqrt{(x_{t1} - w_{c1})^2 + (x_{t2} - w_{c2})^2} < \epsilon_{min} \quad (2)$$

Where w_{c1} and w_{c2} are the coordinates of the closest neuron to the training point x_t shown for the two dimensional case. This updating algorithm moves the neurons near to the training point, wheres $\alpha(e)$ is chosen arbitrarily high at the beginning of the process to provide a fast movement of the mesh.

At the start of the learning process the radius of the neighborhood is choosen to be $R = 2$. Therefore also neurons close to the winning one are effected and move in the direction of the training point. This movement may produce overlapping elements and this problem is discussed in the next section. The radius of the neighborhood decreases as the epochs are repeated. This reflects the idea, that the rough shape of the net is found with a larger radius while the final tuning needs the radius $R = 0$, i.e. only the winning neuron.

B. Edge and Element Treatment

Because the training points lie within the region to be meshed, the learning process results in a mesh, which lacks neurons on the border of the mesh region [5]. To avoid this effect the neurons on the edges in the network could be fixed [6], i.e. they can not move at all. But this prevents the mesh from shifting appropriately in order to reflect the given density of training data. Better results are obtained in this paper by allowing the neurons to move along the border but not into the interior area. This is implemented in the updating formula by determining the coordinates x_{ck} on the specific outline with minimum distance to the training point. Then x_{ck} is taken in place of x_{tk} in (1).

During the learning process groups of these borderline neurons move towards the corners of the bordering polygon and get stuck. A new algorithm concerning the corners is added, which allows the neurons close to corners of the boundary to jump to the adjoining edge. Fig. 3 shows this case, where the neuron n_3 on the edge chooses the line closest to the training point T_p to move on using the algorithm explained above. To preserve the predefined geometry the neighboring neuron n_2 on the opposite line jumps to the corner.

The usage of Kohonen's updating rule (1) without any restrictions can result immediately in a mesh containing overlapping elements, because all nodes in the neighborhood move directly towards the training point regardless of the topology. While the learning process is running, the neural net tries to find a stable state without any topological error. The investigations revealed that there is no guarantee that the minimization process results in an error free mesh. In other words, the primary condition of this minimization problem, i.e. producing topologically correct meshes, is not satisfied when a best solution is found. Using only the stopping criterion (2) may finally result in incorrect meshes. To avoid this topological problem a neuron is updated only if this does not produce any

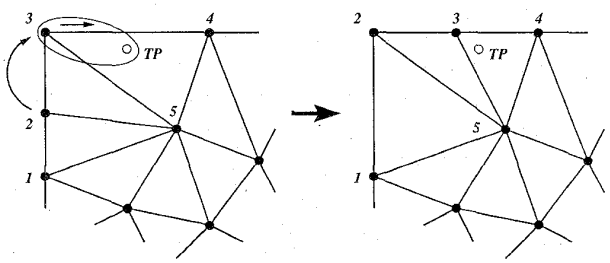

Fig. 3. Edge treatment: a) before and b) after update

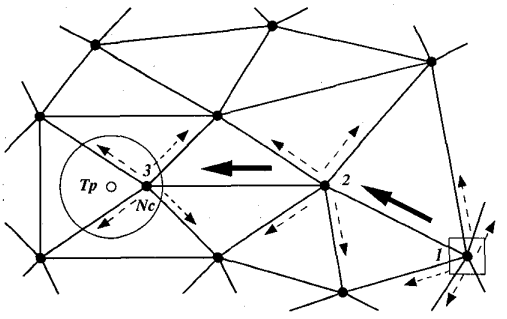

Fig. 4. The search algorithm (start: ⊡, trainweight: T_p, final search area ⊙, closest neuron: N_c, search path: $1, 2, 3$)

overlapping elements, so the process handles correct solutions only and (2) can be used as stopping rule.

III. The Searching Algorithm

The neuron closest to the training vector has to be found and chosen as the winner. The simplest algorithm for calculating the distances between the training vector and every neuron in order to find the closest one is of order $O(n)$. Because parameter variations result in a linear dependency between the number of training points and neurons, the entire process is of order $O(n^2)$ for an epoch.

To reduce the order, the searching algorithm for the winner is changed to include topological knowledge. Taking the information about the neuron's neighbors into account results in a more efficient search for the winner.

The algorithm starts with a randomly chosen neuron and the distances to the training point of both the neuron and all its neighbors are calculated (fig. 4). The neuron, which is closest to the training point, is chosen as the next neuron in the search path. This continues until a neuron N_c is found, whose neighbors are farther away than the neuron itself.

Note, that unlike Kohonen's approach, the topological information of the mesh can be used because every mesh is topologically correct according to the element treatment explained in the last section.

Subsequent routines handle the case of topological exceptions: The last neuron in the search path N_c is not neccessarily the closest one because another neuron which is not a neighbor of N_c could be closer. The area of possible candidates is the region of a circle around the training point with the radius $\overline{T_p N_c}$. An algorithm starts with the neuron N_c and finds all triangles which are part of the circle. If there is a node of these elements found inside the circle it is chosen as N_c and the algorithm restarts. Otherwise the neuron N_c is finally chosen as the winner.

Each winner is saved as the start neuron for the corresponding training point in the next epoch. The entire method reduces the search to a few steps independent of the number of nodes.

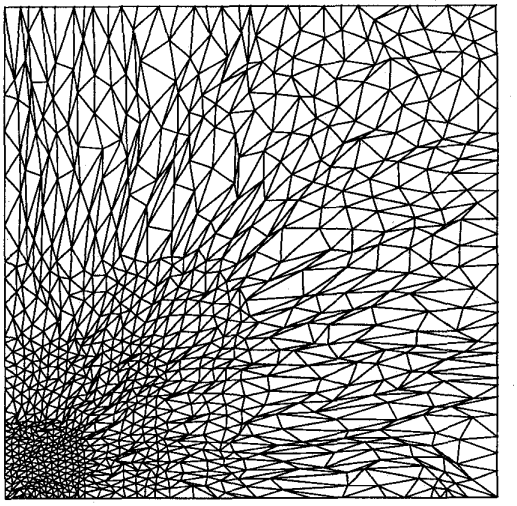

Fig. 5. The final mesh after 113 epochs

IV. Results

Fig. 5 shows the converged self-organizing feature net after 113 epochs when the stopping criterion was reached. The similarity between the element and training set density of fig. 2 is clearly visible. The outer neurons, i.e. nodes, glided along the borderline towards the high density region. Mostly the elements are well shaped, but there are longish ones in the middle, resulting from the fixed topology between the neurons.

Extensive parameter variations concerning the choice of the radius of the neighborhood, the factor $\alpha(e)$ and the number of training nodes have been done. A criterion of comparison is the final sum of Euclidean distances ϵ between each training point and the corresponding closest neuron using (2). The smaller this sum the more simular is the final mesh to the training set. Fig. 6 shows ϵ (related to ϵ_0 of the start mesh) for different numbers of used epochs of $R = 2$ and $R = 1$. It indicates that the error in the mesh density generated compared to that desired is

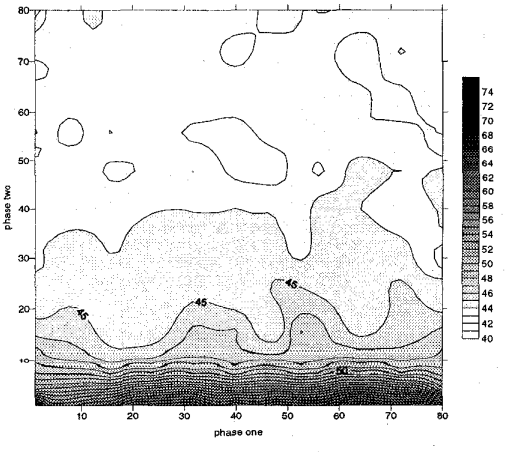

Fig. 6. Sum of minimum Euclidean distances for different number of used epochs for $R = 2$ (abscissa) and $R = 1$ (ordinate)

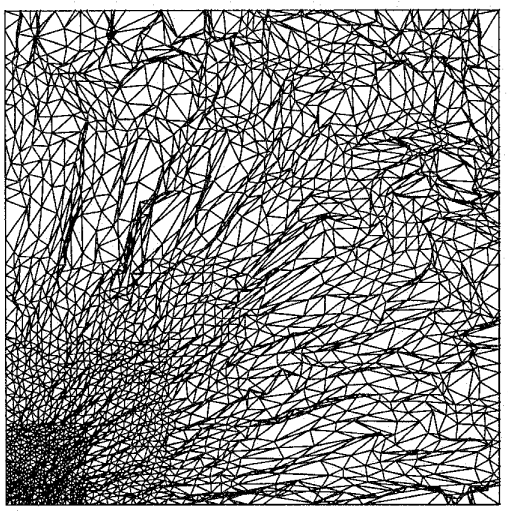

Fig. 7. The mesh with overlapping elements

highly dependent on the second radius ($R = 1$) while the first radius ($R = 2$) has little effect. It was found that the number of training points is best chosen to be twice the number of neurons.

Fig. 7 plots the final mesh produced without the topology check. The visible overlapping elements did not vanish during the updating process.

The method also handles concave regions well because of both the topological check and the special edge treatment. Fig. 8 displays the final mesh of an L-shaped region where the lower right corner was filled stepwise with a high density of training data. Also this mesh reflects the desired element density well. But it reveals the major drawback of this method, which is the fixed topology. Starting with a quite regular mesh in a complex shaped geometry, big differences in the density of the training data satisfy the expections but produce stretched elements. However, these elements could be removed by applying the Delaunay criterion to the mesh in a post-processing pass. An-

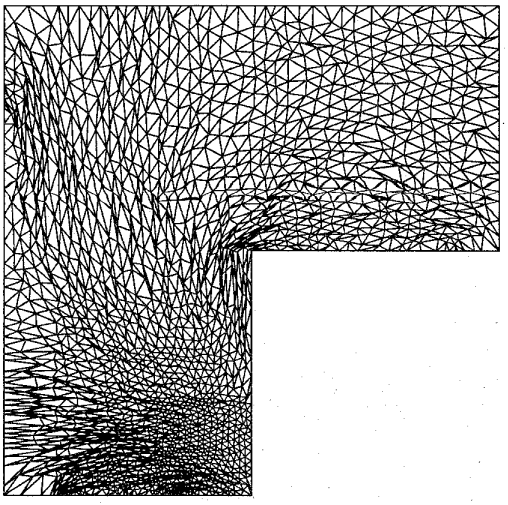

Fig. 8. The final net for a concave region

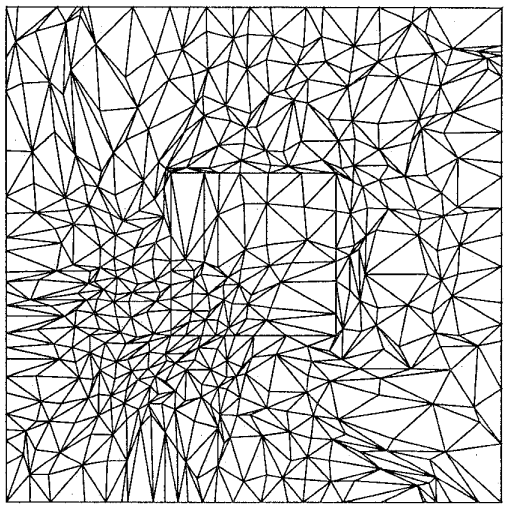

Fig. 9. The final net with 2 regions

other example is shown in fig. 9 where two regions are remeshed simultaneously to a peak of density in the lower left part. The elements can neither move into another region nor is the topology cut to let the elements pass on both sides around the inner region.

Also taking into account that the Kohonen self-organizing feature map needs an initial mesh restricts the potential usage mainly to geometries which can be meshed quickly with a mapping algorithm, or to the field of adaptive remeshing.

V. Conclusions

An improved mesh generation algorithm using self-organizing maps is presented, where the desired element size is given as a density of points in the meshing region. The results show a complexity of almost $O(n)$ for the process. The final meshes reveal a agreement with the training data. Also examples of concave regions are handled well. Although disadvantages are the difficult choice of parameters and the fixed topology.

References

[1] R. Chedid, N. Najjar, "Automatic finite-element mesh generation using artificial neural networks - part I: prediction of mesh density", *IEEE Trans. Magn.*, vol. 32, no 5, pp. 5173-5178, September 1996.

[2] T. Kohonen, "Self-organizing and associative memory", 3rd ed., Springer-Verlag, Berlin, 1989

[3] D.A. Lowther, D.N. Dyck, "A density driven mesh generator guided by a neural network", *IEEE Trans. Magn.*, vol. 29, no 2, pp. 1927-1930, March 1993.

[4] L. Fausett, "Fundamentals of neural netw.", Prentice Hall, 1994

[5] J. Pechoux, D.A. Lowther, "Kohonen maps and automatic mesh generation in two and three dimensions", *Proceedings of the 7th International IGTE Symposium*, Graz, Austria, September 1996, pp. 206-211.

[6] Chang-Hoi Ahn, Sang-Soo Lee, Hyuek-Jae Lee, Soo-Young Lee, "A self-organizing neural network approach for automatic mesh generation", *IEEE Trans. Magn.*, vol. 27, no 5, pp. 4201-4204, September 1991.