A Finite-Element Mesh Generator Based on Growing Neural Networks

Dimitris G. Triantafyllidis and Dimitris P. Labridis, *Senior Member, IEEE*

***Abstract*—A mesh generator for the production of high-quality finite-element meshes is being proposed. The mesh generator uses an artificial neural network, which grows during the training process in order to adapt itself to a prespecified probability distribution. The initial mesh is a constrained Delaunay triangulation (CDT) of the domain to be triangulated. Two new algorithms to accelerate the location of the best matching unit are introduced. The mesh generator has been found able to produce meshes of high quality in a number of classic cases examined and is highly suited for problems where the mesh density vector can be calculated in advance.**

***Index Terms*—Automatic mesh generation, best matching unit location, finite-element method (FEM), let-it-grow (LIG) neural networks, mesh density prediction.**

I. Introduction

THE FINITE-element method (FEM) is one of the most widely used numerical methods in engineering, especially due to its ability to cope with problems of high geometrical complexity, where an analytical solution may be hard to derive. The first step in the FEM is to create a mesh that describes the solving region. Once the mesh is available, a solution for the problem may be derived, by applying the corresponding initial and boundary conditions. The accuracy and the computation time of the solution depend highly on the quality of the initial mesh provided.

A. Common Methods of Mesh Generation

The usual approach is to start the solving process by using an initial coarse mesh and to refine it afterwards by means of an adaptive meshing procedure [1], [2]. During this procedure, the solution error is estimated for each element and the elements with error exceeding a given threshold are split into smaller ones. This way, the initial mesh is refined and the procedure is repeated, until the needed accuracy is met. This technique will provide very accurate results, but may become quite time and memory consuming. Also it does not take into account the experience that may have already been obtained by solving similar problems. In such cases, an experienced user may be able to foresee the density the final mesh should have. It is, therefore, within the aim of the present paper to provide a mesh generator based on growing neural networks that is highly suited for the parametrical analysis of problems with similar geometrical definitions. Starting from an initial coarse mesh, the proposed mesh generator will provide a quality mesh, which will serve as a good starting point for the adaptive refinement to follow.

One of the most common methods of mesh generation is to start from an infinite triangular grid, which is then superimposed on the object to be meshed. The elements that fall outside the object are either removed or trimmed to fit the geometry of the object. This method will produce very-high-quality elements in the interior of the meshing area, but is not suitable in cases where small objects are present in a large solving region. Such problems appear in overhead power transmission line (OTL) problems, where the size of the conductors is minimal compared to the solving region. The initial grid would have to be made of elements of very small size in order to match the geometry of the conductors. This would produce an unnecessary large number of elements in the rest of the solving area. A hybrid method could probably be used in which the area to be meshed would be subdivided in smaller regions, but the method would be case specific and might not be able to cope with other problems it was not designed for.

Another popular method of gridding is to start from an initial (Delaunay) triangulation of the geometry of the problem and then smooth the initial set of nodes by using a method such as Laplacian smoothing. This method will also produce very high-quality meshes, but does not take into account experience that may have been gained by triangulating similar geometries in the past.

B. Recent Developments in Mesh Generation Using ANN

In order to address the problem of triangulating geometries which present similarities, the use of backpropagation artificial neural networks (ANNs) has been proposed for predicting the mesh density vector of certain electromagnetic field problems [3]–[5]. The derived mesh density vector may then be used to create the mesh, which maintains the mesh density properties of the prediction. Various mesh generators have been proposed for this purpose.

A Delaunay-based density driven mesh generator, guided by an ANN providing the mesh density vector, may be used for the generation of the mesh [5], [6]. Also, the utilization of self-organizing maps (SOMs) [7] has been proposed for the same purpose [8], [9]. A probability distribution (pd) is used to adapt the coordinates of an initial constant topology of triangles or rectangles. This approach suffers from the fact that the topology of the mesh is set *a priori*, making it difficult to adapt itself in cases were the pd presents high irregularities. In these cases the mesh appears to be "stretched" and the elements are of poor quality. In addition, all the nodes located on the boundaries have

Manuscript received August 31, 2000; revised December 12, 2001.

The authors are with the Power Systems Laboratory, Department of Electrical and Computer Engineering, Aristotle University of Thessaloniki, GR-54124 Thessaloniki, Greece (e-mail: dimtri@egnatia.ee.auth.gr; labridis@eng.auth.gr).

Digital Object Identifier 10.1109/TNN.2002.804223

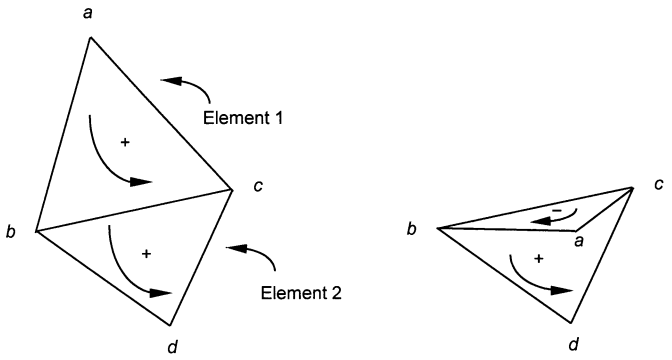


Fig. 1. Node a has been moved over edge b–c, causing an inconsistency in the mesh. The nodes of element 1 are originally stored in the following order—(a, b, c). In the inconsistent mesh, the order has been reversed—(a, c, b).

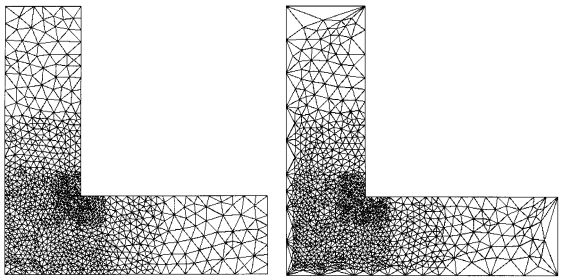


Fig. 2. In mesh (a) the creation of nodes on the boundaries has been favored, but not in mesh (b).

to be set in advance [8], thus, the process can not be totally automated.

Recently, a mesh generator, based on let-it-grow (LIG) ANNs has been presented [10]–[12]. In this approach, an initial coarse mesh is used to generate the final mesh. The initial mesh is also used to define the solving region of the problem, as well as to carry the information of the piecewise constant probability density function (pdf) of the pd. This means that in regions were a finer refinement is necessary the initial mesh has also to be finer, thus, user intervention is not totally avoided, which is a drawback if the mesh generation process should be entirely automated. In addition, another disadvantage of the LIG mesh generator is highlighted in [12], namely, the location of the best matching unit. The proposed algorithms in [12] do not allow the generation of large meshes due to their high computational complexity. The pdf in this approach may as well be derived by means of a backpropagation ANN [3]–[5].

C. Aim of this Paper

The main scope of this paper is to explore whether an LIG mesh generator may be applied successfully to the parametrical salvation of a large number of cases presenting similar geometry. In this aspect, human intervention should be minimized at all steps of the mesh generation procedure. Also, the mesh generator must provide consistent (conforming) quality meshes at an acceptable time, which will be used as a good starting point for an adaptive meshing procedure, instead of having to start the meshing procedure each time from scratch.

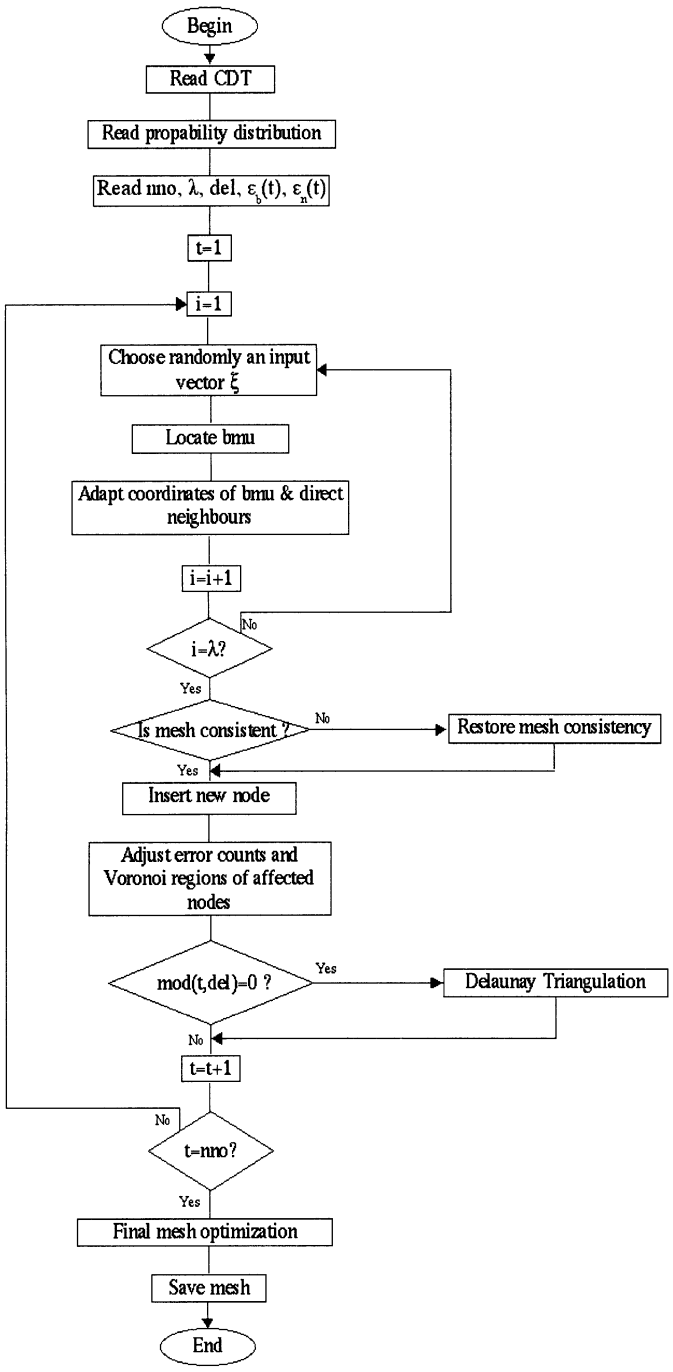


Fig. 3. Flowchart of the proposed mesh generator.

This paper presents a new version of an LIG ANN-based mesh generator for the production of two-dimensional (2-D) meshes. The mesh generation starts using as an initial mesh a constrained Delaunay triangulation (CDT), which may be derived automatically using an appropriate algorithm [13]. This is a great advantage, because the meshing procedure does not depend on the quality of the initial mesh, which will be refined only where necessary. Also, human intervention is avoided, since the initial mesh is provided automatically. A pd, with a pdf that is not necessary piecewise constant, as was originally proposed in [10], may then be used to drive the mesh generator to produce the final mesh, maintaining

the mesh density properties of the pd. The proposed mesh generator also introduces two new algorithms, which reduce the computational complexity of locating the best matching unit. As a result, the mesh generation process is accelerated, allowing for the generation of large meshes. Additionally, the mesh generator offers a guarantee that the produced mesh is conforming.

II. Mesh Density Prediction

The mesh generator has been devised in such way that it is independent of the technique that provides the mesh density vector, which should be considered as an input to it. The derivation of the mesh density vector is actually a case-specific problem. For specific problems in the magnetic device area, the application of ANNs has been proposed to automate this process [3], [4]. Furthermore, in this paper we apply the technique proposed in [5], which is suitable for problems in the transmission line area.

In the method described in [5], an ANN is trained using the backpropagation method. The inputs to the ANN are the height h of the conductor above the air–ground boundary, the distance r from the conductor to the point where the mesh density is to be estimated and the frequency to earth resistivity ratio f/ρ. f is the frequency of the current flowing through the conductor and ρ is the resistivity of the ground.

Once the ANN has been trained it can produce a mesh density prediction for new cases, which are presented to it for the first time in minimal time. The output of the ANN is a number corresponding to the mesh density inside a square (called magnifier in [4]), which along with adjacent squares covers the whole solving region. According to this prediction a uniform cloud of points on the plane is produced, confined within the corresponding square. The union of the cloud of points for all squares produces the final cloud of points, which drives the mesh generator. A thorough presentation of this method may be found in [5].

III. Mesh Generator

The inputs to the proposed mesh generator are:

1) CDT of the problem to be meshed;
2) pd $p(\xi)$, $\xi \in R^2$;
3) number of elements nel or the number of nodes nno to be created;
4) two additional user-defined constants λ and del and two user defined functions $\varepsilon_b(t)$, $\varepsilon_n(t)$, where t is the number of nodes that have been inserted into the mesh, during the mesh growing procedure.

Typical values for λ, del, $\varepsilon_b(t)$, and $\varepsilon_n(t)$ will be given in Section VI. The CDT may be readily obtained by using a variety of algorithms available [13], [14]. $p(\xi)$ is given in the form of a 2-D cloud of points, which is denser where a finer mesh is required. $p(\xi)$ may be automatically obtained by using an ANN [3]–[5] or may be defined by the user [10].

The algorithm is very close to the original LIG algorithm [15] with some variations, which are necessary for the mesh generation problem. The LIG ANN consists of neurons (nodes), which are connected to form a triangular mesh A on the plane.

TABLE I
For Each Input Vector, the Node Number of the BMU Located at the Previous Search That Took Place for the Same Input Vector Is Stored in the Extended Cache Array

Input vector ξ_i		Node number of last located bmu, corresponding to input vector ξ_i
ξ_1	$\rightarrow$	145
ξ_2	$\rightarrow$	1035
$\vdots$	$\vdots$	$\vdots$
$\xi_{nsamples}$	$\rightarrow$	3

Every neuron c is associated with a position vector $w_c \in R^2$, which contains the coordinates of the neuron on the plane. An input vector $\xi \in R^2$ is then selected randomly according to the pd $p(\xi)$. The node of the mesh that is closest to ξ, called the best matching unit (BMU), is selected. For this purpose, the Euclidean metric has been used

$$|w_{\mathrm{BMU}} - \xi| = \min_{i \in A} |w_i - \xi| . \tag{1}$$

A signal counter T_c is assigned to each node c. The signal counter for the BMU is increased by one if the node lies in the interior of the solving region. However, if the BMU lies on a boundary or a material interface, the signal counter is increased by 1.5. This is done to favor nodes that lie on boundaries or interfaces, as to create a sufficient number of nodes on them.

The coordinates of the BMU are adapted according to

$$w_{\mathrm{BMU}}^{\mathrm{new}} = w_{\mathrm{BMU}}^{\mathrm{old}} + \varepsilon_b(t) \cdot (\xi - w_{\mathrm{BMU}}^{\mathrm{old}}). \tag{2}$$

Also, the coordinates of all direct neighbors of the BMU are adapted

$$w_i^{\mathrm{new}} = w_i^{\mathrm{old}} + \varepsilon_n(t) \cdot (\xi - w_i^{\mathrm{old}}) \qquad \forall i \in N_{\mathrm{BMU}} \tag{3}$$

where N_{BMU} denotes the direct topological neighbors of BMU. In some cases, the functions $\varepsilon_b(t)$ and $\varepsilon_n(t)$ may be selected to be constant, but if the starting mesh is a CDT it is best to choose small values for $\varepsilon_b(t)$ and $\varepsilon_n(t)$ in the beginning of the meshing procedure and increase their values as the mesh grows. This may be explained as follows: the coordinate adaptation resulting from (2) and (3) is proportional to the distance between the input vector ξ and the BMU or its neighbors. In a sparse mesh, this distance is rather large compared to a dense mesh. Therefore, the coordinate adaptation may lead to mesh inconsistencies, by moving a node over the edge of the triangle it belongs to. Although this error is corrected afterwards by the proposed mesh generator, it can be avoided by using not constant $\varepsilon_b(t)$ and $\varepsilon_n(t)$ functions.

If a node c lies on a boundary or an interface of two regions, then the adaptation takes place only for the component of w_c^{new} which is parallel to the line defined by the boundary or interface. If a node lies on the intersection of boundaries and/or interfaces,

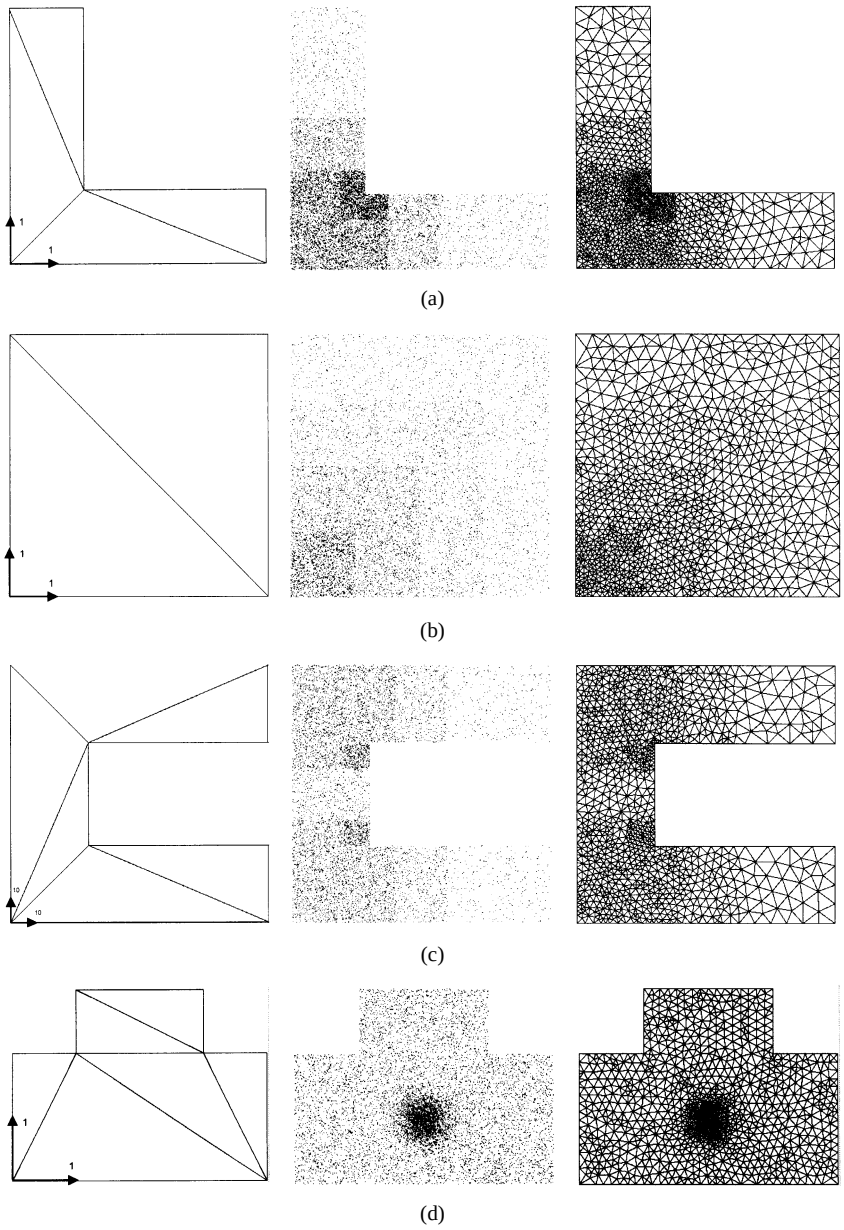


Fig. 4. Test cases (a)–(d). Left: corresponding CDT; middle: probability distribution used; right: final mesh produced.

then no adaptation takes place. The above procedure is repeated λ times.

Before the algorithm can continue with the insertion of a new node, the consistency of the mesh has to be checked. This is very important, since during the adaptation process some nodes may have been moved over an edge of a triangle, causing some triangles to overlap others (Fig. 1). The problem may be confronted by taking care so that all triangles have the same orientation during the mesh generation process. In the proposed work all triangles of the initial CDT have a counterclockwise orientation of their nodes, thus an overlapping of triangles will cause a triangle to appear with opposite orientation. In this case, the corresponding node's position vector is set back to the original value it had before the adaptation process took place.

Once mesh consistency has been ensured, a new node is inserted on the edge of an existing triangle. The node q having the highest signal counter is selected. The outgoing edge from q to its farther direct neighbor f is split in the middle. This way, two new triangles are created if the edge lies in the interior of the solving region or one new triangle is created if the edge lies on a boundary. The new node r is inserted in the middle of edge q–f, its position vector given by

$$w_r = \tfrac{1}{2}\left(w_q + w_f\right). \tag{4}$$

Circular boundaries are treated as follows: a circle is described by its center, radius, and a polygon to which the circle is circumscribed. If node r is to be inserted on edge q–f, which belongs to the polygon, then it is inserted in the middle of the arc q–f instead. Further coordinate adjustment of this node is not allowed.

Finally, a signal counter T_r has to be assigned to the new node r. Also the signal counters of its neighbors have to be redefined. This is necessary, because if r was present in the mesh from the

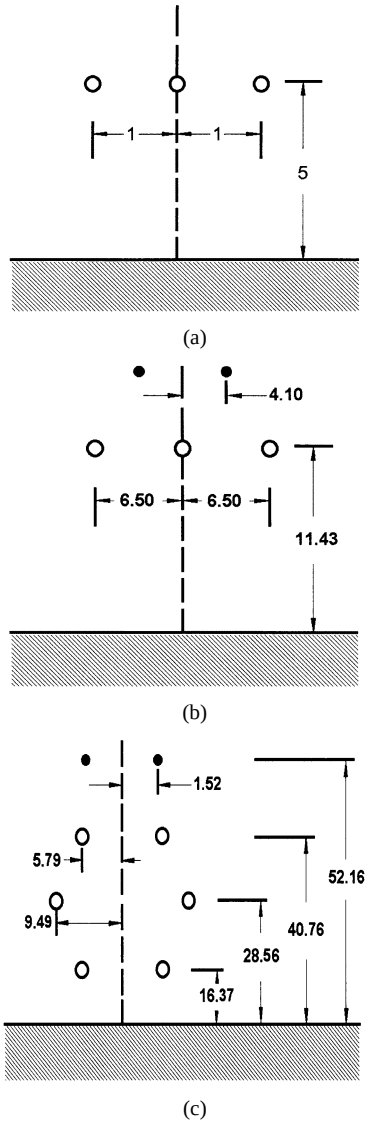


Fig. 5. Overhead transmission lines geometrical configurations of test cases (e) to (g).

beginning it would have been BMU for a number of times, instead of its neighbors. Therefore, the neighbors' signal counters have to be decreased. For all the direct topological neighbors $i \in N_r$ of r, their signal counters are decreased according to

$$\tau_i^{\text{new}} = \tau_i^{\text{old}} - \Delta\tau_i \tag{5a}$$

$$\Delta\tau_i = \frac{S_i^{\text{old}} - S_i^{\text{new}}}{S_i^{\text{old}}} \cdot \tau_i^{\text{old}} \tag{5b}$$

while the new node r receives the total amount of signal counters subtracted from its neighbors

$$\tau_r = \sum_{i \in N_r} \Delta\tau_i. \tag{6}$$

S_i denotes the area of the Voronoi region of node i. For simplicity, S_i has been approximated according to

$$S_i \cong l_i^2, \quad \text{where } l_i = \frac{1}{|N_i|} \sum_{k \in N_i} |w_i - w_k| \tag{7}$$

TABLE II
PROBABILITY DISTRIBUTIONS FOR TEST CASE (d)

Type	Number of samples	Extensions	Center	Standard deviation
Uniform	7500	All	-	-
Gaussian	2500	-	(2,1)	1

where $|N_i|$ is the number of the direct topological neighbors of node i.

Furthermore, to improve mesh quality on interfaces and boundaries if r lies on an interface or boundary, its signal counter is multiplied by a factor of 1.05, otherwise it is multiplied by 0.95. The same applies to the neighbors N_r of r. Fig. 2 demonstrates the advisability of this proposal. Factors 1.05 and 0.95 have been determined by numerical experiments and are most suitable for the category of problems examined in this paper. A Delaunay triangulation takes place after del nodes have been inserted in the mesh in order to improve the quality of the mesh.

The procedure is repeated until the required number of elements nel or number of nodes nno is reached. At last, a final optimization takes place by moving each node that does not lie on a boundary or interface of the mesh, toward the centroid of the polygon, formed by its neighboring nodes [16]. Also, a last Delaunay triangulation takes place. Fig. 3 shows a flow chart of the algorithm.

The mesh generator has been implemented using the Library of Efficient Data Structures and Algorithms (LEDA), a publicly available collection of C++ classes for combinatorial and geometric computing [17]. LEDA provides a useful set of data structures, highly suitable for the development of the presented mesh generator. A data type called *graph* has been used, which allowed the usage of *node maps* and *edge maps*. This way for every node its direct neighbors were readily available, as well as all its adjacent edges. Additional information was stored on nodes and edges, such as the triangle and region it belongs to, etc.

IV. LOCATING THE BMU

For a mesh with a large number of nodes, the procedure of locating the BMU may be extremely time consuming. For this purpose, the following technique of gradient search has been implemented [12].

1) Start from an arbitrary node c in the mesh.
2) Calculate the distance between the input vector ξ and node c.
3) Calculate the distance between ξ and each of the direct topological neighbors of c.
4) Compare the distances of Step 3 to the distance of Step 2. Keep the smallest distance and set c to be the node corresponding to it.
5) Continue from Step 2, until no neighbor of c is closer to ξ than c itself.

This approach of locating the BMU has been improved even further by creating an appropriate set of starting nodes for the search. This was implemented by maintaining some sort of cache memory, which contains a small, but representative, set of the nodes of the whole mesh. For this purpose, an amount

TABLE III
DATA ON TEXT CASES (e)–(g)

	20 kV single circuit line (Test case e)	150kV single circuit line (Test case f)	735kV double circuit line (Test case g)
Phase conductors.			
Conductor type	Solid	ACSR	ACSR
Outside diameter (mm)	5.6419	25.146	35.103
Inside diameter (mm)		9.71	23.364
dc resistance (Ω/km)	0.2833	0.09136	0.04965
Ground wires.			
Conductor type	-	solid St conductor	Alumoweld strand
Outside diameter (mm)		0.9525	0.9779
dc resistance (Ω/km)		3.4431	1.4913
f/ρ ratio (Hz/Ωm)	1	100	1000
f (Hz)	1	100	1000

TABLE IV
QUALITY FACTORS OF THE TEST CASES (a)–(d)

Case and Short Description	Number of nodes	Number of elements	CPU Time (sec)	Minimum quality factor	Maximum quality factor	Mean quality factor	Joint quality factor
(a):L	1106	2069	17.29	0.431403	0.999971	0.929726	0.913173
(b):Square	1104	2100	16.4	0.337652	0.999999	0.928945	0.915841
(c):C	1108	2051	16.13	0.293024	0.999995	0.927746	0.901394
(d):T	1108	2101	16.4	0.376328	0.999973	0.929496	0.899764

TABLE V
QUALITY FACTORS OF THE TEST CASES TAKEN FROM [10]

Description of case taken from [10]	Number of nodes	Minimum quality factor	Maximum quality factor	Mean quality factor
L shaped domain	648	0.389	1.0	0.925
Square shaped domain	1125	0.28	1.0	0.921

of 1.5% of the number of nodes constituting the mesh, which have been BMU at some point in the algorithm, are kept in an array named Cache. Step 1 of the procedure is then replaced by

1) Search the Cache array and locate the node, which is closest to ξ. Set this node to be c.

The remaining steps are kept the same.

The Cache technique produced similar results, concerning the time consumed in locating the best matching unit as the one presented in [12], where variable start nodes have been used. In [12], at the beginning of the algorithm, the triangle E_ξ of the initial coarse mesh is located in which each input vector ξ lies. When ξ is presented to the LIG ANN, one of the nodes of E_ξ are used as a starting point for the BMU location.

To improve the Cache technique even further, another approach has been introduced, implementing an *extended Cache* array. The extended Cache array is actually a lookup table of constant size, equal to the number of input vectors. For each input vector ξ, the last located BMU, which corresponded to ξ, is stored in the array in the form of a pair of integers (id of ξ, id of BMU), as illustrated in Table I. The symbol id refers to the integer numbering of the input vectors and the nodes of the mesh. The next time the same input vector ξ is presented to the LIG ANN, the extended Cache array is being looked up and the BMU stored in pair with ξ is used as a starting point for the BMU location. Again, the gradient-search algorithm is used from this point on. It is reasonable to expect that the new BMU will be near to the last located BMU. For the initialization of the extended Cache array, the absolute BMU for each input vector ξ may be located once at the beginning of the algorithm. Numerical experiments presented in Section VI show that this method greatly improves the performance of the algorithm.

It should be noted that the extended Cache algorithm in its presented form works properly only if all the input vectors are known at the start. This does not introduce any drawbacks in the algorithm as long as the mesh density prediction takes place in advance, before the mesh generation begins. The proposed mesh generator has been designed under the assumption that no new input vectors are presented to it during mesh generation. We believe that it should be a simple task to change the algorithm of the mesh generator in order to cope with cases, where the input vectors are not known at start. In this case, the extended Cache array would be a lookup table of dynamical size.

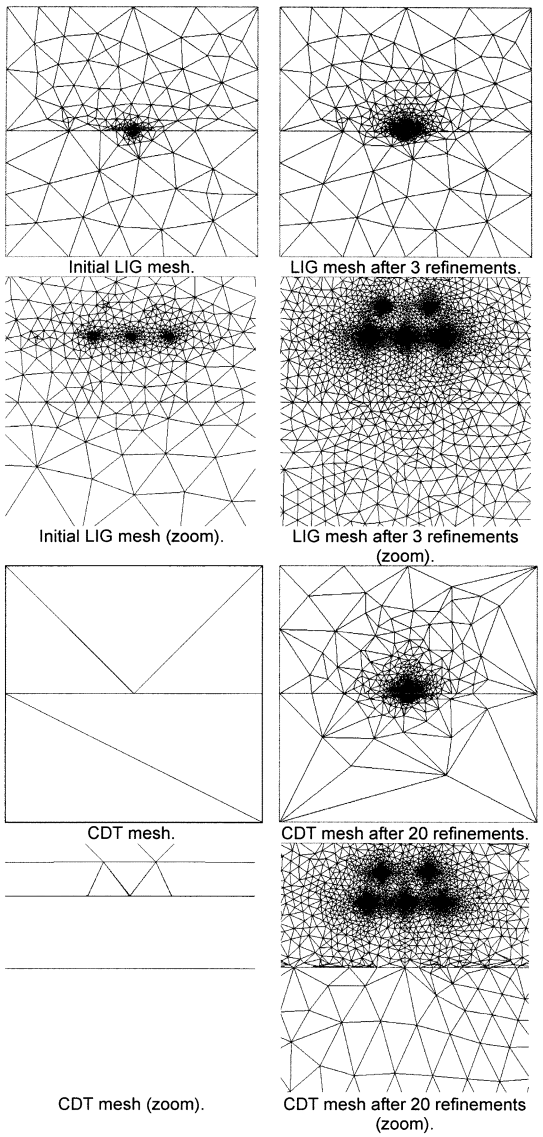


Fig. 6. Comparison between meshes produced by the proposed mesh generator and CDT meshes for test case (f).

V. Computational Complexity

In this section, the computational complexity of the BMU location methods presented in Section IV is analyzed. For completeness, the absolute BMU detection method is presented as well.

A. Absolute BMU Detection

At a given instant, let N be the number of nodes in the mesh. In order to locate the BMU N, distance comparisons have to take place, calculating the distance of the input vector ξ and all the nodes in the mesh. Assume that the distance calculation and comparison for one node is of constant time α. Then, for N nodes, αN time is required. This procedure is repeated λ times, during which the number of nodes N remains constant. Thus, for N nodes a total amount of time equal to $\lambda\alpha N$ is necessary. The node insertion process will insert one node each time, thus the number of nodes during the next BMU location will be $(N+1)$. The total amount of time spent for the BMU location is now $\lambda\alpha(N+1)$. The amount of time spent from the beginning of the algorithm, with N_1 number of nodes to a given instant with N_2 number of nodes, is equal to

$$\begin{aligned}\sum_{i=N_1}^{N_2}\lambda\alpha i &= \lambda\alpha\left[N_1+(N_1+1)+(N_1+2)+\cdots+N_2)\right]\\ &= \lambda\alpha\left[N_1+(N_1+1)+(N_1+2)+\cdots+(N_1+v)\right]\\ &= \lambda\alpha\left[(v+1)N_1+(1+2+\cdots+v)\right]\\ &= \lambda\alpha\left[(v+1)N_1+\frac{v+v^2}{2}\right]\\ &= \lambda\alpha\left[(N_2-N_1+1)N_1+\frac{N_2-N_1}{2}+\frac{(N_2-N_1)^2}{2}\right]\end{aligned} \tag{8}$$

where $v = N_2 - N_1$.

Thus, the algorithm for the absolute BMU detection is of N^2 complexity.

B. Gradient Search BMU Detection

By using the gradient search algorithm the amount of time spent for the BMU location reduces significantly, because at each detection it is not necessary to make distance comparisons between the input vector and all the nodes of the mesh. The procedure operates by moving from one node to the neighboring node, until no neighbor is closer to the input vector than the current node. The number of iterations necessary obviously depends on the mesh topology and cannot be estimated for the general case. Nevertheless, it is reasonable to assume that the larger the mesh becomes the more distance comparison have to take place, since the starting point for the search is set at random. Assuming that the number of distance comparisons is proportional to the size of the mesh, thus equal to kN, where $0 < k < 1$, the amount of time spent can be derived in a similar fashion as in (8)

$$\Delta t(N_1, N_2) = k\lambda\alpha \cdot\left[(N_2-N_1+1)N_1+\frac{N_2-N_1}{2}+\frac{(N_2-N_1)^2}{2}\right]. \tag{9}$$

The gradient search algorithm is also of N^2 complexity, but due to the factor k, the BMU location time reduces significantly.

C. Variable Start Node Algorithm

The variable start node algorithm uses the gradient search algorithm, but each BMU location does not start from a random node of the mesh but from a node, which belongs to a triangle of a copy of the initial mesh. Since the initial mesh does not evolve along with the LIG NN, when the mesh has grown significantly there is a large number of nodes N_e inside of each of the triangles e of the copy of the initial mesh. As the mesh continues to grow, N_e continues to grow as well and will follow the mesh density vector.

Most BMU locations will have to take place at locations where mesh density is higher. This also happens to be where N_e is highest, because N_e depends on the mesh density vector. A simplifying assumption to exemplify the behavior of the algorithm would be to assume that $N_e = mN$, $0 < m < 1$.

TABLE VI
COMPARATIVE RESULTS FOR TEST CASES (e)–(g)

Test case (e)					
	EMTP	CDT (15^{th} iteration)	% Difference	LIG (4^{th} iteration)	% Difference
Nodes		8003		8480	5.96
CPU Time [sec]		479.2		232	-51.59
Averaged quality		0.85300		0.86482	1.386
Joint quality		0.81187		0.83207	2.488
$\bar{V}$ (V/m)	0.285+j0.0305	0.299+j0.0297	4.83	0.291+j0.0303	2.0744

Test case (f)					
	EMTP	CDT (21^{st} iteration)	% Difference	LIG (4^{th} iteration)	% Difference
Nodes		8496		8487	-0.11
CPU Time [sec]		784.2		255	-67.48
Averaged quality		0.83680		0.85898	2.651
Joint quality		0.08487		0.80924	853.505
$\bar{V}$ (V/m)	0.408+j1.575	0.407+j1.57	-0.313	0.410+j1.576	0.0904

Test case (g)					
	EMTP	CDT (18^{th} iteration)	% Difference	LIG (4^{th} iteration)	% Difference
Nodes		6517		7125	9.33
CPU Time [sec]		705.2		217	-69.23
Averaged quality		0.80132		0.83924	4.732
Joint quality		0.29230		0.58780	101.095
$\bar{V}$ (V/m)	0.712+j13.116	0.733+j12.911	-1.549	0.746+j12.606	-3.862

This assumption holds true for a uniform mesh density distribution and will lead to results, which are in favor of the variable start node algorithm. In this case, each BMU location is confined within a submesh of size mN. Since the gradient search algorithm is applied, only kmN distance calculations have to take place. Therefore,

$$\Delta t(N_1, N_2)km\lambda\alpha = \cdot\left[(N_2-N_1+1)N_1+\frac{N_2-N_1}{2}+\frac{(N_2-N_1)^2}{2}\right]. \quad (10)$$

The computational complexity of the variable start node algorithm is also N^2. BMU location time reduces because $k < 1$ and $m < 1$.

D. Cache Algorithm

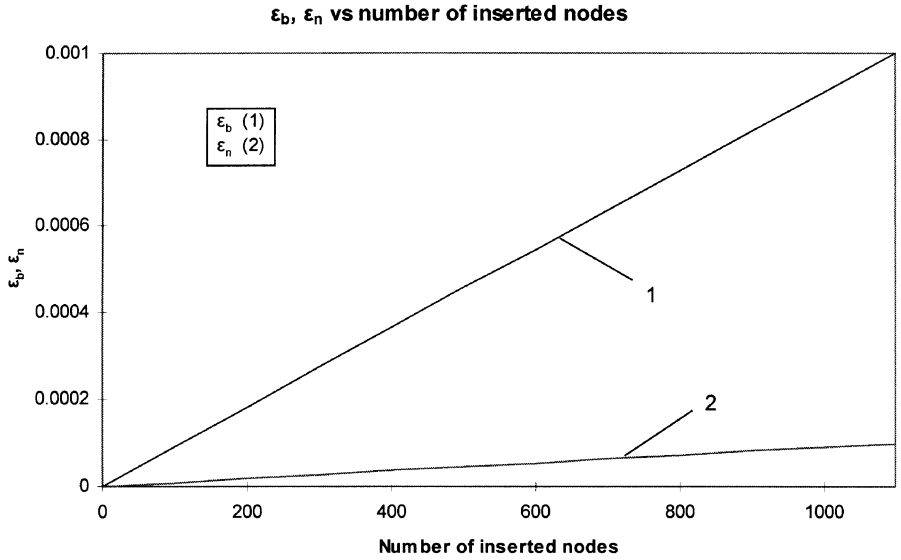
The Cache algorithm also uses the gradient search algorithm. In this case, the procedure starts from a node stored in the Cache array and not from a random node. The size of the Cache array increases proportional to the mesh size and can be set equal to sN, $0 < s < 1$. The search algorithm starts by selecting the node from the Cache array, which is closest to the input vector ξ. In order to find the start node within the Cache array, sN distance comparisons have to take place, consuming αsN computational time.

Once the starting node has been located, the gradient search algorithm is applied in order to locate the BMU. The new BMU will usually be near the starting node, which is an older BMU located in a case where an old input vector was in the region of the current input vector. Therefore, only a small number of distance comparisons is required to locate the new BMU. Since the search area for the BMU is concentrated around the starting node, and the number of starting nodes grows along with the mesh, it is expected that the number of distance comparisons to locate the BMU is independent of the size of the mesh. Assuming a mean value of n distance comparisons the computational time required is αn, therefore constant.

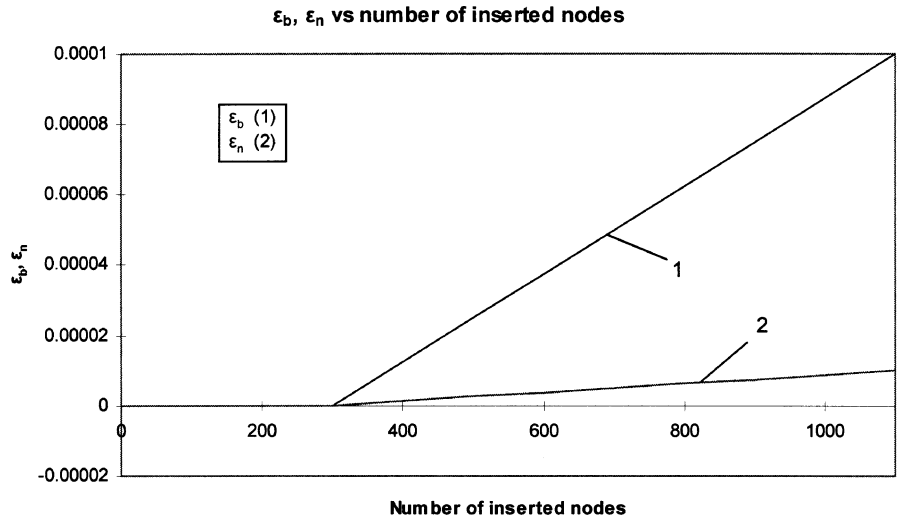
The total computational time is given by

$$\begin{aligned}\Delta t(N_1, N_2) &= \sum_{i=N_1}^{N_2} \lambda\alpha si + \sum_{i=N_1}^{N_2} \lambda\alpha n \\ &= \lambda\alpha s\left[(N_2-N_1+1)N_1+\frac{N_2-N_1}{2}+\frac{(N_2-N_1)^2}{2}\right] \\ &\quad + (N_2-N_1+1)\lambda\alpha n. \end{aligned} \quad (11)$$

The complexity of this algorithm is also N^2, due to the fact that the size of the Cache array grows with the mesh.



(a)



(b)

Fig. 7. (a) $\varepsilon_b(t)$ and $\varepsilon_n(t)$ for test cases (a)–(d). (b) $\varepsilon_b(t)$ and $\varepsilon_n(t)$ for test cases (e)–(g).

E. Extended Cache Algorithm

In this case, the extended Cache array constitutes a lookup table of constant size. The amount of time spent to look the table up is minimal, constant, and may be ignored. Once the starting node has been located from the extended Cache array, the gradient search algorithm is applied. The algorithm starts from an old BMU, which was recorded the last time, the same input vector ξ, as in the current case, was presented to the algorithm.

For the same reasons as in the Cache array algorithm the number of distance comparisons required does not depend on the mesh size and may be assumed equal to n. Thus, the computational time required is

$$\Delta t(N_1, N_2) = \sum_{i=N_1}^{N_2} \lambda\alpha n = \lambda\alpha n(N_2 - N_1 + 1). \qquad (12)$$

The extended Cache algorithm has N complexity and is highly suited for generating large meshes.

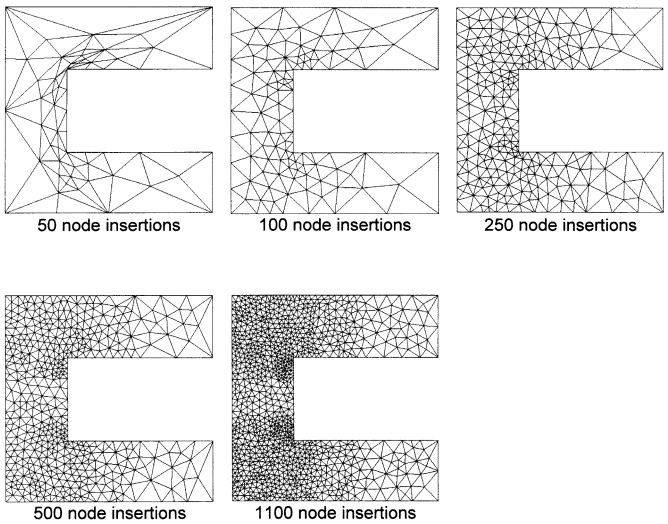
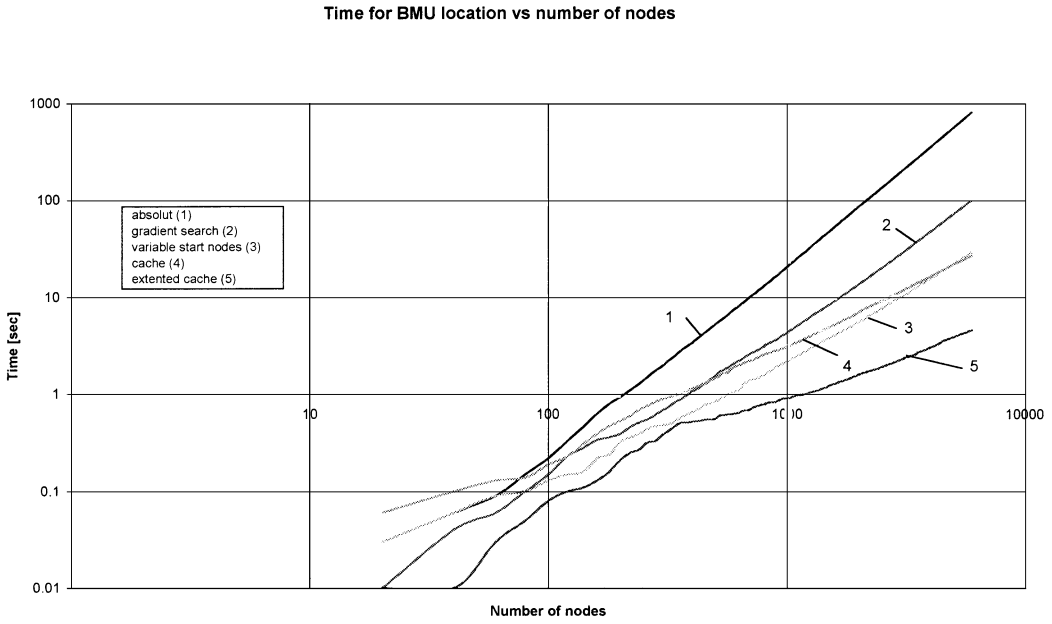
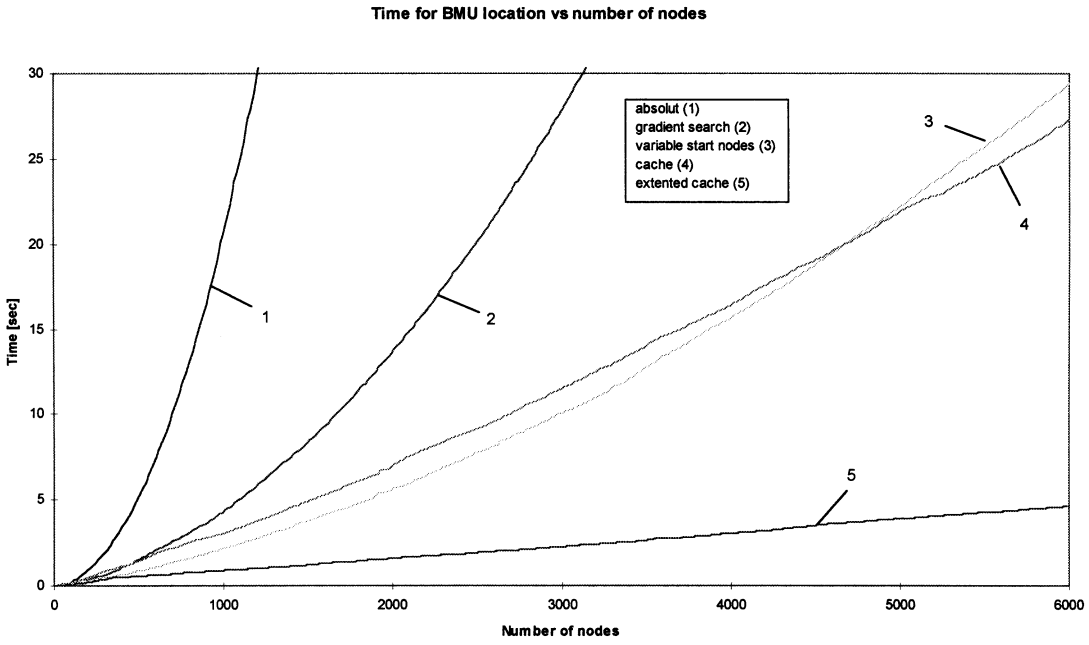


Fig. 8. Evolution of mesh into its final form for test case (c).



(a)



(b)

Fig. 9. (a) Number of nodes versus time for test case (c) (see Fig. 4). (b) Linear plot of (a).

The complexity analysis presented here will be highlighted by experimental results presented in the next section.

VI. Results

Seven test cases have been considered to present the behavior of the proposed mesh generator, as depicted in Figs. 4 and 5. Test cases (a) and (b) are an L-shaped domain and a square, respectively. The geometrical configurations of the domains as well as the corresponding probability distributions have been taken from [10] and are examined here as a comparison study. Test case (c) is a C-shaped core taken from [4]. The pd has been arbitrary defined by the authors. In cases (a)–(c), the probability distributions are piecewise defined within square regions, as can be noticed from the respective drawings in the middle of Fig. 4. These test cases have been selected as a means of comparison with results obtained in [10]. To demonstrate the independence between the initial mesh and the pd used, a uniform distribution of points has been superimposed to a Gaussian distribution of points (Table II), in test case (d). This case constitutes a reverse T slot embedded conductor [18].

Finally, three additional cases (e)–(g) are shown in Fig. 5. These cases present three different cases of OTL problems located in an area 10 km $\times$ 10 km. The dimensions of the conductors and related data are presented in Table III. The pd for these cases has been produced by means of an ANN [5], as explained

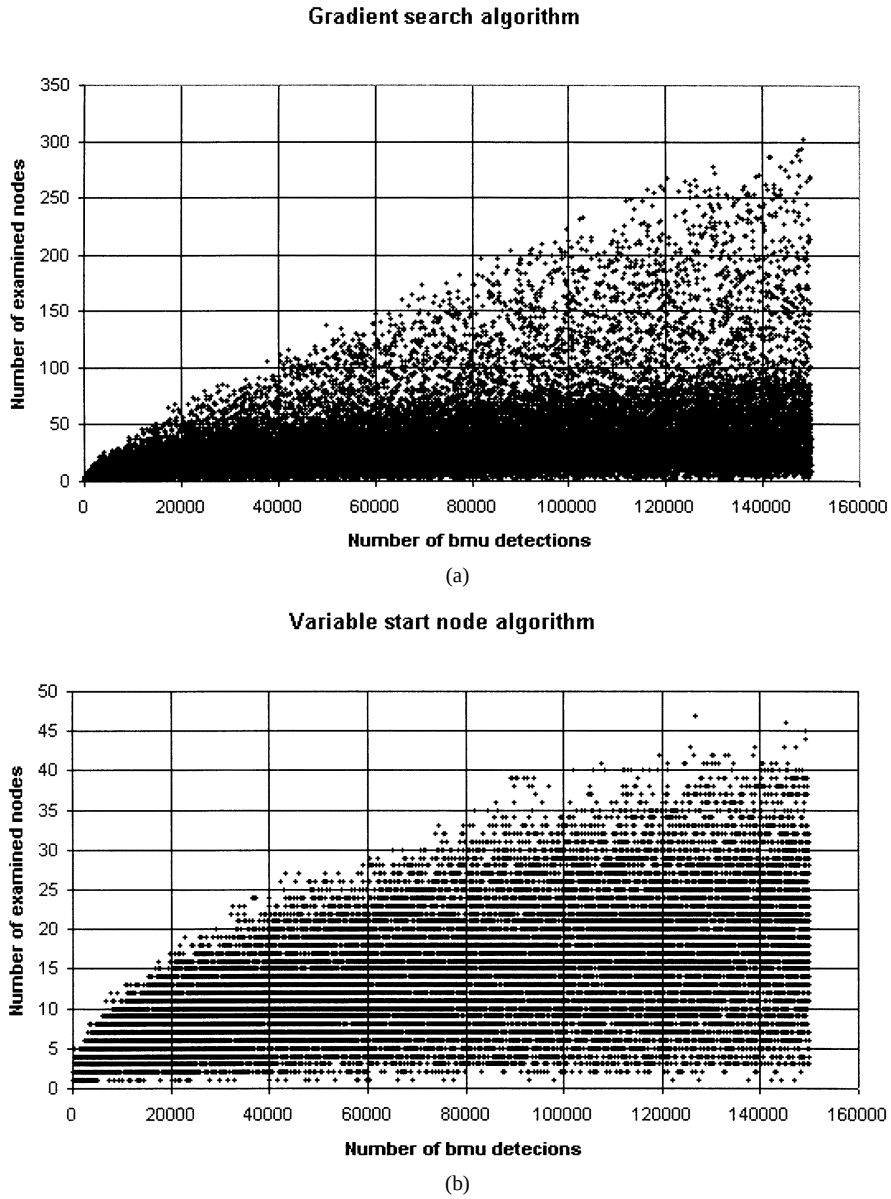


Fig. 10. Number of nodes examined until the BMU is located for (a) the gradient search algorithm and (b) the variable start node algorithm.

in Section II. The problems belong to a family of problems with similar geometry, for which the ANN has been trained to predict the mesh density. The examples also demonstrate the capability of the proposed mesh generator to cope with cases where small features are present in a large solving region and high irregularities are present in the input pd.

Table IV presents the number of nodes and elements for test cases (a) to (d), as well as the CPU time for the mesh generation on a Pentium based machine at 200 MHz, with 64 MB of RAM, running a Linux operating system. In addition, the mean quality of the mesh of the triangles for each test case, as well as the worst, best and joint quality factors obtained are presented. The quality of a triangle i, having sides of length a, b, c has been defined as [20]

$$q_i = \frac{8(s-a)(s-b)(s-c)}{abc} \tag{13a}$$

$$s = \frac{a+b+c}{2}. \tag{13b}$$

To calculate the mean quality factor q_t of the whole mesh, the weighted mean of q_i has been used

$$q_t = \frac{\sum_{i=1}^{\text{nel}} S_i \cdot q_i}{\sum_{i=1}^{\text{nel}} S_i} \tag{14}$$

where S_i is the area of triangle i and nel is the total number of triangles in the mesh.

The joint quality factor q_j of the mesh has been calculated according to [20]

$$\frac{1}{q_j} = \frac{1}{\text{nel}} \sum_{i=1}^{\text{nel}} \frac{1}{q_i}. \tag{15}$$

Table V reproduces the results from [10] for the test cases of the L- and square-shaped domains. The proposed mesh generator produces similar results as the one proposed in [10]. It has

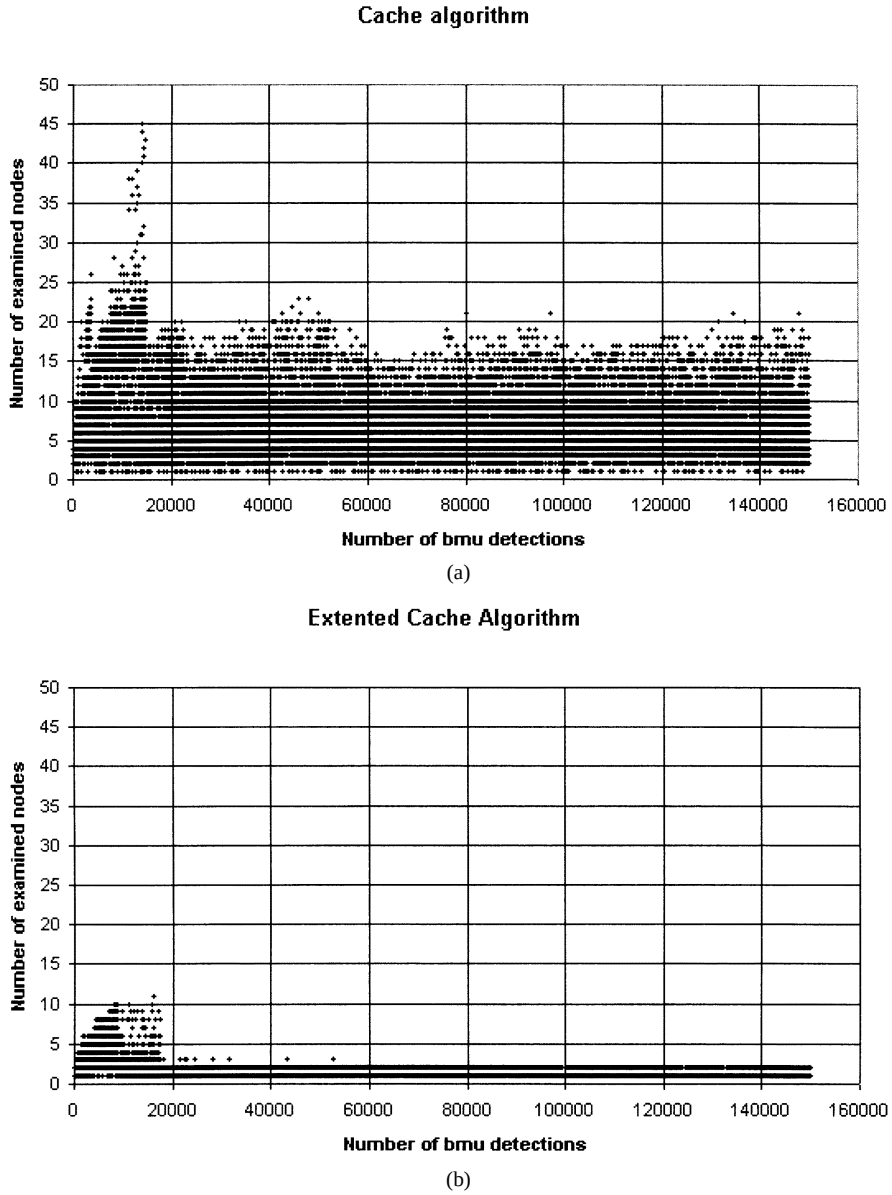


Fig. 11. Number of nodes examined until the BMU is located for (a) the Cache algorithm and (b) the extended Cache algorithm.

a slight advantage in the mean quality factor and performs quite better, concerning the minimum quality factor.

Fig. 6 presents meshes from test case (f) produced by the proposed mesh generator in comparison with meshes obtained by an adaptive meshing procedure starting from an CDT. The mesh produced by the mesh generator had to be refined three times in order to produce the same results as the CDT mesh, which had to be refined 20 times. The complex voltage drop along the upper left conductor of each test case has been calculated using the FEM [19]. For this reason, a zero sequence system of currents, having a magnitude of 1000 A per phase was applied to each OTL. In test case (g), only the left half of the line was energized. As a means of reference, the voltage drop has also been calculated by the electromagnetic transients program (EMTP) [22]. Results are presented in Table VI. The CPU time presented includes mesh generation. A reduction of CPU time from 50% up to 70% has been recorded, which is very significant if a large number of similar problems are to be solved.

In all test cases, the user-defined constants λ and del have been set equal to $\lambda = 25$ and del $= 20$. The user defined functions $\varepsilon_b(t)$ and $\varepsilon_n(t)$ are depicted in Fig. 7. These values have been selected after experimenting with a variety of test cases and have been found to be most suitable for the generation of high-quality FEM meshes.

In order to present how the mesh develops into its final form, Fig. 8 presents the C-shaped domain after 50, 100, 250, 500, and 1100 nodes have been inserted into the initial CDT mesh. It becomes apparent that even at the beginning of the procedure, high-quality meshes are produced, which follow the prescribed mesh density vector.

Test case (c) has been probed further, in order to investigate the amount of time spent for the BMU location, as well as the influence of erroneous BMU detection on the stability of the algorithm. The mesh generator was asked to produce a final mesh by adding 6000 nodes to the existing nodes of the initial CDT mesh. In the first case, the absolute BMU was located using (1)

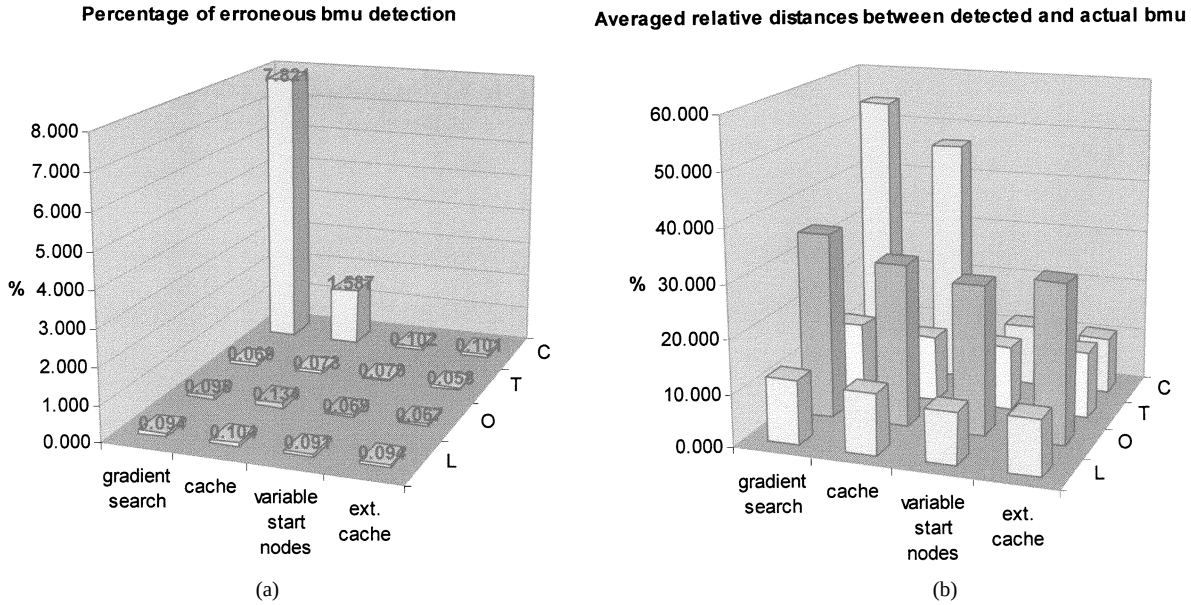


Fig. 12. (a) Percentage of erroneous BMU detection for test cases (a)–(d) (L, O, T, C) and four detection algorithms. (b) Averaged relative distances between detected and actual BMU.

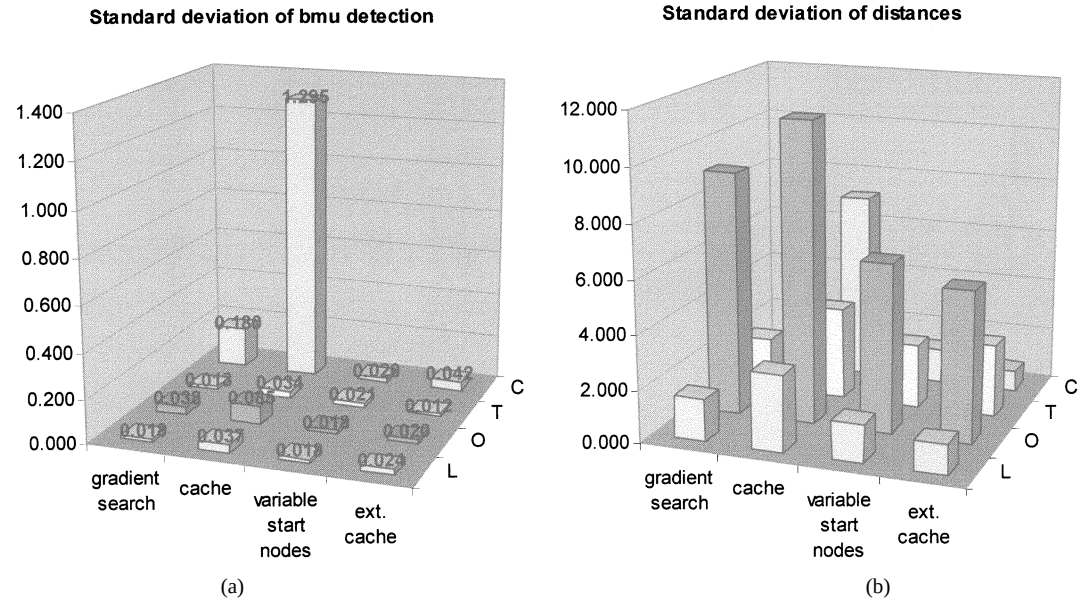


Fig. 13. (a) Standard deviation for the BMU detection percentages presented in Fig. 12(a). (b) Standard deviation for the averaged relative distances presented in Fig. 12(b).

(Fig. 9, curve 1). Then, the techniques presented in Section IV were used. Curve 2 represents the gradient search algorithm, curve 3 the variable start nodes technique from [12], curve 4 the Cache array method, and finally, curve 5 the extended Cache array technique.

The first two techniques are obviously unacceptable for the production of large meshes. The variable start nodes technique presented in [12] greatly reduces the BMU location time, although the Cache array method appears to outperform it after a critical point, which in the presented case is near 5000 nodes. The fastest method, as expected, is the extended Cache array method.

The variable start node, the Cache and the extended Cache algorithms utilize the gradient search algorithm by issuing a good starting point to begin the search for the BMU. Each method introduces a different approach, but all three expect that the actual BMU will be near the starting point. A number of nodes are examined, until no neighboring node is nearer to the input vector than the current node examined. The number of nodes that have to be examined until the algorithm comes to a stop will highlight the behavior of each algorithm. Therefore, this number has been recorded as a function of the number of BMU detections having taken place for test case (c). For 6000 node insertions, $\lambda \cdot 6000 = 25 \cdot 6000 = 150\,000$ BMU detections took place. This way, four graphs were generated depicted in Figs. 10 and 11.

As expected, the number of examined nodes increases as the mesh grows in the case of the gradient search algorithm

[Fig. 10(a)] and in the case of the variable start node algorithm [Fig. 10(b)]. This is in agreement with the computational complexity analysis of Section V. The number of nodes examined in the case of the Cache algorithm [Fig. 11(b)] stabilizes after a short peak at the beginning of the mesh growing procedure. According to this graph the algorithm should be of N complexity. Still, time spent to locate the starting node from within the Cache array increases as the mesh grows, because the size of the array increases as well, as explained in Section IV. In the case of the extended Cache algorithm, the number of nodes examined stabilizes at 1 or 2 after a short peak at the beginning. This means that the BMU is located at most at a direct neighbor of the starting node. Both algorithms rely on the discipline that LG NN organize themselves in such a fashion that topologically close elements in the network should have similar input vectors mapped onto them and vice versa [21]. Thus, the short peak present in both cases may be explained by the fact that in the beginning the NN has not yet adapted itself perfectly to the pd.

The utilization of the above techniques introduces a proportion of error in the location of the BMU, since the absolute BMU may not be found in some cases, especially in nonconvex regions [12]. To explore the introduced error, test cases (a) to (d) have been considered. The proposed mesh generator was asked to produce a mesh of additional 3000 nodes to the nodes already present in the CDT, thus $\lambda \cdot 3000 = 25 \cdot 3000 = 75\,000$ BMU locations took place in each case. The percentage of failing to locate the absolute BMU has been recorded and the numerical experiment was repeated until the recorded percentage would stabilize at a constant value. From 10 to 25 repetitions were necessary varying from case to case. Results are presented in Fig. 12(a), along with the standard deviation σ of the recorded erroneous BMU location percentages from the presented average [Fig. 13(a)]. The gradient search algorithm produced the worst results. The Cache array method, on the other hand, produced better results, but with a great deviation from the mean value. The variable start nodes and the extended Cache array techniques produced the best results. The percentage of erroneous BMU location is at worst 0.1%, while deviation is minimal.

Additionally, for the cases where erroneous BMUs were located, the relative averaged distance between the absolute and the detected BMU has been recorded and presented in a similar fashion in Fig. 12(b), along with the corresponding standard deviation [Fig. 13(b)]. The distances have been normalized to the length of each domain. The variable start node algorithm and the extended Cache algorithm again performed better than the other methods. The highest averaged relative distance of about 50% was recorded in test case (c) when the gradient search algorithm was applied.

In order to exemplify the influence of erroneous BMU detection on the output of the mesh generator test case (e) has been triangulated under the precondition that the percentage of erroneous BMU detections is, successively, 100%, 50%, and 0%. The corresponding meshes are presented in the left column of Fig. 14. The right column zooms on the right conductor of each case. Apparently, the algorithm is quite insensitive to the percentage of erroneous BMU detections, since even at 50% error the resulting mesh preserves the mesh density vector, although not as close as in the case of 0% error. As expected, in the case of 100% the resulting mesh follows a random mesh density vector. Obviously, results depend on the input mesh density vector. The influence of erroneous BMU detection will become higher as the concentration of the input signals is in a certain area of the domain, compared to the rest of the domain, becomes higher as well. Therefore, the example presented here can qualify as a worst-case scenario, since the majority of the input vectors is concentrated around the conductors. Bearing this in mind, the recorded percentages of erroneous BMU detection presented in Fig. 12(a) should be considered to have minimal influence on the output of the mesh generator.

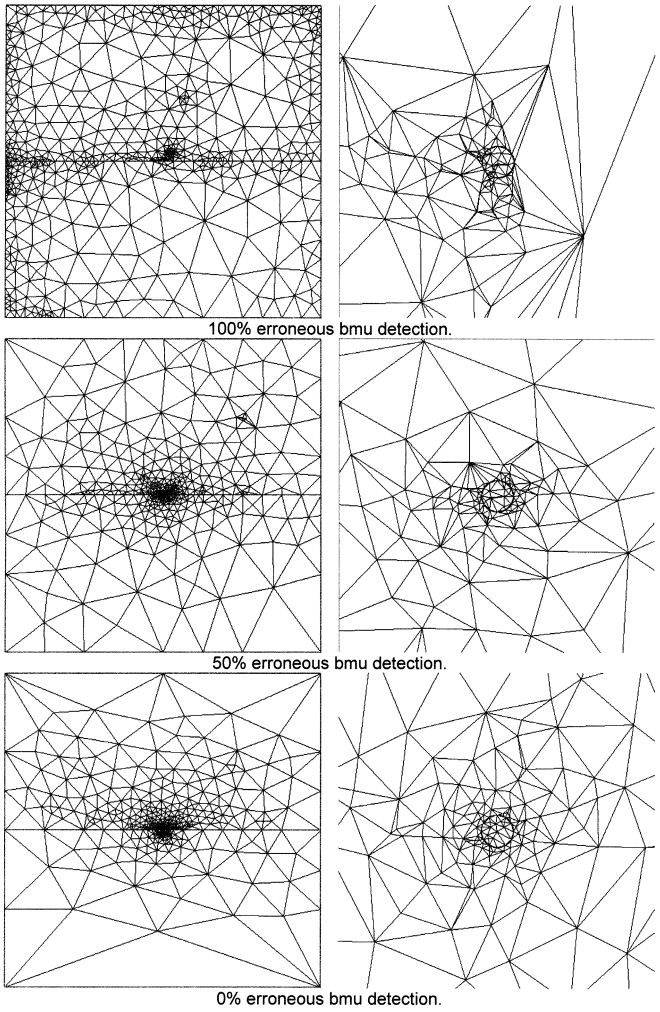


Fig. 14. Left column: Overall view of the mesh produced by the mesh generator for test case (e) for various predefined percentages of erroneous BMU detections. Right column: Zoom onto the right conductor for the corresponding mesh presented on the left.

VII. Conclusion

A mesh generator based on LIG ANNs has been proposed. The mesh generator produces a triangular mesh according to a probability distribution of points on the plane. The initial mesh is a CDT of the domain to be triangulated. The mesh generator has been tested to a number of cases and was able to produce good-quality meshes. Two techniques for accelerating

the mesh generator have been introduced and their properties have been investigated. One of the proposed techniques reduces the computational complexity of locating the BMU to N. The proposed mesh generator may be used in addition to a mesh density prediction technique, in order to fully automate the mesh generation process. The Cache and the extended Cache technique for accelerating the training process may be applied to LIG ANNs, regardless of the application, hence, not only for mesh generation purposes.

REFERENCES

[1] Z. J. Cendes and D. N. Shenton, "Adaptive mesh refinement in the finite element computation of magnetic fields," *IEEE Trans. Magn.*, vol. 21, pp. 1811–1816, Sept. 1985.

[2] N. A. Golias and T. D. Tsiboukis, "An approach to refining three-dimensional tetrahedral meshes based on Delaunay transformations," *Int. J. Numer. Methods Eng.*, vol. 37, pp. 793–812, 1994.

[3] D. N. Dyck and D. A. Lowther, "Determining an approximate finite element mesh density using neural network techniques," *IEEE Trans. Magn.*, vol. 28, pp. 1767–1770, Mar. 1992.

[4] R. Chedid and N. Najjar, "Automatic finite-element mesh generation using artificial neural networks—Part I: Prediction of mesh density," *IEEE Trans. Magn.*, vol. 32, pp. 5173–5178, Sept. 1996.

[5] D. G. Triantafyllidis and D. P. Labridis, "An automatic mesh generator for handling small features in open boundary power transmission line problems using artificial neural networks," *Commun. Numer. Methods Eng.*, vol. 16, no. 3, pp. 177–190, Mar. 2000.

[6] D. A. Lowther and D. N. Dyck, "A density driven mesh generator guided by a neural network," *IEEE Trans. Magn.*, vol. 29, pp. 1927–1930, Mar. 1993.

[7] T. Kohonen, "The self-organizing map," *Proc. IEEE*, vol. 78, pp. 1464–1480, Sept. 1990.

[8] C.-H. Ahn *et al.*, "A self-organizing neural network approach for automatic mesh generation," *IEEE Trans. Magn.*, vol. 27, pp. 4201–4204, Sept. 1991.

[9] B.-S. Jeong, S.-Y. Lee, and C.-H. Ahn, "Automatic mesh generator based on self-organizing finite-element tessellation for electromagnetic field problems," *IEEE Trans. Magn.*, vol. 31, pp. 1757–1760, May 1995.

[10] S. Alfonzetti *et al.*, "Automatic mesh generation by the let-it-grow neural network," *IEEE Trans. Magn.*, vol. 32, pp. 1349–1352, May 1996.

[11] S. Alfonzetti, "A finite element mesh generator based on an adaptive neural network," *IEEE Trans. Magn.*, vol. 34, pp. 3363–3366, Sept. 1998.

[12] ——, "High-quality finite element mesh generation by means of an artificial neural network," in *Proc. 7th Int. IGTE Symp.*, 1996, pp. 177–182.

[13] J. R. Shewchuk, "Triangle: Engineering a 2D quality mesh generator and Delaunay triangulator," in *Proc. 1st Workshop Appl. Comput. Geometry*. Philadelphia, PA: Assoc. Comput. Machinery, May 1996, pp. 124–133.

[14] L. P. Chew, "Constrained Delaunay triangulations," *Algorithmica*, vol. 4, pp. 97–108, 1989.

[15] B. Fritzke, "Wachsende Zellstrukturen - ein selbstorganizierendes neuronales Netzwerkmodell," Ph.D. dissertation, University of Erlangen-Nürnberg, Germany, 1992.

[16] N. Golias and T. Tsiboukis, "Adaptive refinement in 2-D finite element applications," *Int. J. Numer. Mod., Elect. Dev. Fields*, vol. 4, pp. 81–95, 1991.

[17] K. Mehlhorn and S. Näher, "Algorithm design and software libraries: Recent developments in the LEDA project," in *Algorithms, Software, Architectures, Information Processing*. Amsterdam, The Netherlands: Elsevier, 1992, vol. 92, pp. 493–505.

[18] P. Hammond and T. D. Tsiboukis, "Dual finite-element calculations for static electric and magnetic fields," *Proc. Inst. Elect. Eng.*, pt. A, vol. 130, no. 3, pp. 105–111, 1983.

[19] D. P. Labridis and P. S. Dokopoulos, "Electromagnetic forces in three-phase rigid busbars with rectangular cross-sections," *IEEE Trans. Power Delivery*, vol. 11, pp. 793–800, Apr. 1996.

[20] D. A. Lindholm, "Automatic triangular mesh generation on surfaces of polyhedra," *IEEE Trans. Magn.*, vol. 19, pp. 2539–2542, Nov. 1983.

[21] B. Fritzke, "Growing cell structures—A self-organizing network for unsupervised and supervised learning," *Neural Networks*, vol. 7, no. 2, pp. 1441–1460.

[22] H. W. Dommel, *Electromagnetic Transients Program Reference Manual*. Portland, OR: Boneville Power Administration, 1986.

Dimitris G. Triantafyllidis was born in Stuttgart, Germany, on September 25, 1972. He received the Dipl. Eng. and Ph.D. degrees from the Department of Electrical and Computer Engineering, Aristotle University of Thessaloniki, Thessaloniki, Greece, in 1996 and 2001, respectively.

He is with the Department of Electrical and Computer Engineering, Aristotle University of Thessaloniki. His research interests include finite elements, power systems engineering, and neural networks.

Dr. Triantafyllidis is a Member of the Society of Professional Engineers of Greece.

Dimitris P. Labridis (S'88–M'90–SM'00) was born in Thessaloniki, Greece, on July 26, 1958. He received the Dipl.-Eng. and the Ph.D. degrees from the Department of Electrical and Computer Engineering, Aristotle University of Thessaloniki, Thessaloniki, Greece, in 1981 and 1989, respectively.

Since 1982, he has been with the Department of Electrical and Computer Engineering at the Aristotle University of Thessaloniki, first as a Research Assistant, later as a Lecturer, and then as an Assistant Professor and, since 2001, as an Associate Professor. His special interests are power system analysis with special emphasis on the simulation of transmission and distribution systems, electromagnetic and thermal field analysis, and artificial intelligence applications in power systems and power-line communications.