ELSEVIER

Expert Systems with Applications 29 (2005) 193–206

Expert Systems with Applications

www.elsevier.com/locate/eswa

An ANN-based element extraction method for automatic mesh generation

S. Yao, B. Yan, B. Chen, Y. Zeng*

Concordia Institute for Information Systems Engineering, Concordia University, 1455 de Maisonneuve West, CB-410-16, Montreal, Que., Canada H3G 1M8

Abstract

This paper proposes an artificial neural network based element extraction method for automatic finite element mesh generation. A finite element mesh is a discretized representation of a geometric domain. A domain is discretized into elements, which may be triangles, quadrilaterals, tetrahedron, or hexahedron. The element extraction method repeatedly generates element (s) within the domain using some predefined 'if-then' rules until the whole domain is filled with required elements. This method has an advantage of creating meshes for domains with complex boundaries. However, the 'if-then' rules for element extraction are usually difficult to acquire, because these rules not only generate elements but also change the problem they are designed to solve. In this paper, a Back-Propagation (BP) neural network is used to represent the 'if-then' element extraction rules and to train the relationship behind these rules. The input for this network includes the coordinates of some boundary points while the output defines the parameters for extracting an element. In order to generate good quality element while keeping the updated problem still solvable, the design and definition of the neural network is more complex than those in the traditional classification problems. This paper discusses issues related to the design of the neural network for element extraction. Numerical experiments on quadrilateral mesh generation have shown that this method is effective.

Keywords: Mesh generation; Element extraction; Artificial neural network; Quadrilateral mesh; Evolution of rules

1. Introduction

Over the last four decades, automatic mesh generation has drawn a great deal of attention from academic research and industrial applications. Its benefit to many areas including structural analysis and optimization, computational fluid dynamics, and geometric modeling has kept it an active research (Almeida, 2000; Canales, Tárrago, & Hernándeztopic, 1994; Lira, 2002). Because of its importance and underlying challenges, many mesh generation methods and algorithms have been developed (Brown, 1981; Cavendish Field, & Frey, 1985; Sadek, 1980; Woo, 1984; Yerry & Shephard, 1984). As a result, commercial software packages for automatic mesh generation are widely available in the market. However, most of the currently available mesh generation methods are heuristic and can be applied in solving only some specific problems. The mesh generation of geometric shapes with complex boundary is still an open question (Shewchuk, 2002).

Element extraction is recognized as a flexible method to process geometric shapes with complex boundaries (Saxena & Perucchio, 1990). The meshes generated by this method usually have high quality elements in the area close to the geometric boundary. This is a feature, which is critical for complex engineering problems Cavalcante Net, 2001. However, its drawback is also serious. The high quality boundary elements are usually followed by poor elements in the middle of the geometric domain, mainly because the robust rules for element extraction are extremely difficult to acquire. This drawback has made it difficult to be widely accepted as a robust mesh generation method.

Based on a mathematical analysis of the problem underlying the element extraction method, this paper proposes an ANN-based element extraction method for automatic mesh generation. In this method, the artificial neural network is used to evolve rules for extracting elements. These rules will generate elements of good-enough quality around both the boundary and the middle of the domain.

Among numerous publications in mesh generation, there are only a few addressing the application of artificial neural

* Corresponding author. Tel.: +1 514 848 2424; fax: +1 514 848 3171.
E-mail address: zeng@ciise.concordia.ca (Y. Zeng).

0957-4174/$ - see front matter © 2005 Elsevier Ltd. All rights reserved.
doi:10.1016/j.eswa.2005.01.019

network to mesh generation. Topping, Khan, and Bahreininejad (1997) presented a parallel processing implementation for neural computing and its application to finite element mesh decomposition. Their method is useful in predicting the number of elements that may be generated for the initial background mesh in the subdomain generation method. In another paper, Manevitz, Yousef, and Givoli (1997) applied neural networks in the problem of mesh placement for the finite element method using the self-organizing algorithm of Kohonen. Ahmet and Ahmet (2002) proposed a method for two-dimensional mesh generation by means of feed forward single layer neural networks. These methods belong to other categories of mesh generation methods. Rule acquisition in element extraction remains to be a problem.

For the sake of simplicity, we will use two-dimensional quadrilateral mesh generation as an example. The goal of this problem is to generate a two-dimensional quadrilateral mesh from a predefined piece-wise boundary, satisfying the following requirements:

- inner corners of each element should be between 45 and 135°;
- the aspect ratio (the ratio between opposite edges) and taper ratio (the ratio between adjacent edges) of each element should be between 0.1 and 10;
- the mesh around the short boundary components should be dense and vice versa; and
- the transformation from dense mesh to coarse mesh should be smooth.

The rest of this paper is organized as follows. Section 2 reviews the element extraction method and identifies the nature of underlying problems. Section 3 presents the architecture and algorithm of an ANN-based element extraction method. Section 4 discusses the generation of training samples. Section 5 gives some numerical examples. Section 6 concludes this paper and gives some future research issues.

2. Element extraction method: formulation and problems

2.1. Formulation of element extraction method

Element extraction method generates a finite element mesh by extracting elements along the domain boundary. The element can be a triangle, quadrilateral, tetrahedron, or hexahedron depending on the given mesh generation problem. In this mesh generation process, one element is usually cut from the domain according to a set of predefined 'if-then' rules. Each time an element is removed from the domain, the domain boundary will be updated for further element extraction. This process continues until the domain becomes a valid element. Fig. 1 is an example of element

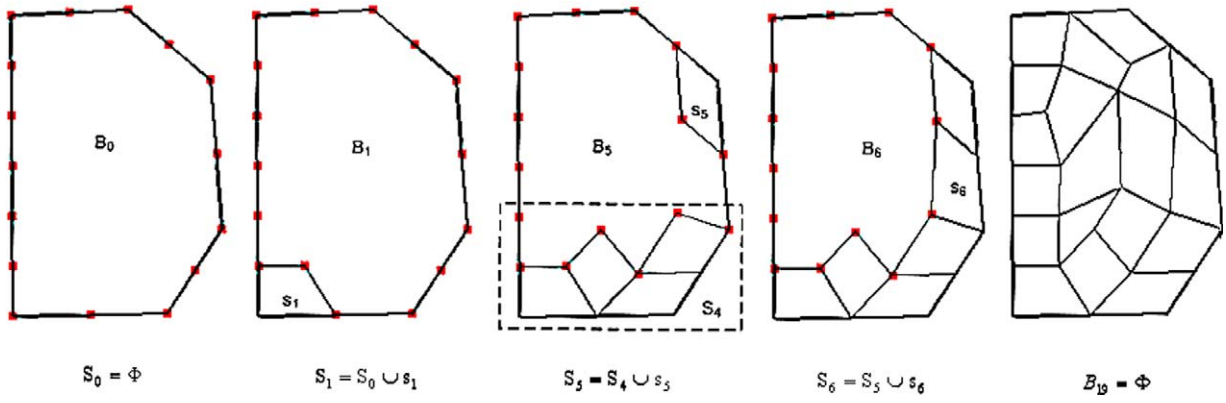

Fig. 1. Quadrilateral mesh generation by element extraction.

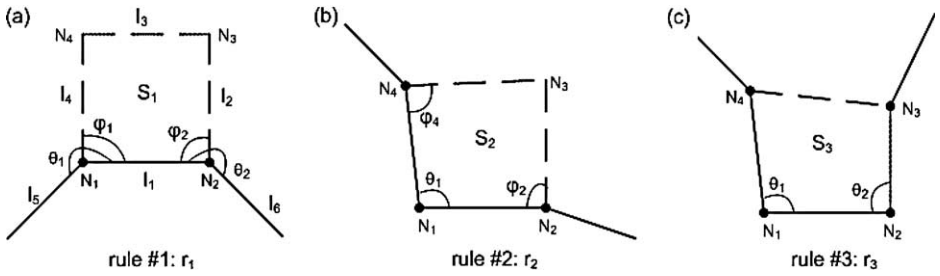

Fig. 2. Three basic element extraction rules.

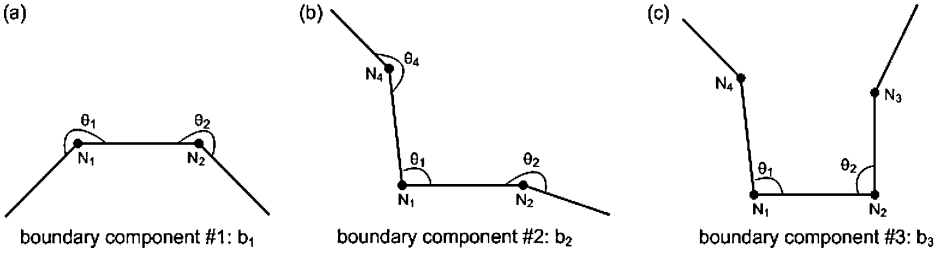

Fig. 3. Basic boundary components.

extraction applied to two-dimensional quadrilateral mesh generation.

Mathematically, this mesh generation process can be described using the following equations:

$$r_j : b_i \rightarrow s_i, \quad S_{i+1} = S_i \cup s_i, \quad B_{i+1} = B_i / s_i, \quad \forall b_i \in B_i \; \exists \, r_j \in R, \tag{1}$$

where r_j is one rule in R, which is a set of if-then rules for extracting an element; b_i is a part of domain boundary B_i; s_i is the element generated around b_i; S_i is the generated finite element mesh; symbols $\cup$ and/are set operations 'union' and 'difference', respectively.

If we take $S_i \cup B_i$ as the state of the mesh generation process, denoted by Ω_i ($\Omega_i = S_i \cup B_i$), then element extraction rules in R can be seen as operators on the state Ω_i. This is represented in the following equation:

$$r_j : \Omega_i \rightarrow \Omega_{i+1}, \quad \forall \, \Omega_i \; \exists \, r_j \in R. \tag{2}$$

For the two-dimensional quadrilateral mesh generation problem, there are three basic rules in R, which are shown in Fig. 2. They extract an element from the domain by adding three, two, or one line(s), respectively.

Corresponding to the three rules in Fig. 2, there are three basic types of boundary components as is shown in Fig. 3. These three types of components define a base for representing any two-dimensional domains in terms of the rules given in Fig. 2. They are called primitive boundary components.

The precedents of these rules are the coordinates of a group of boundary points whereas the conclusions include the number of lines to be added and the parameters defining these lines. The following lists the rules used in a system developed by Zeng and Cheng (1993).

(1) If $\theta_1 \geq U_c$, $\theta_2 \geq U_c$, then $\phi_1 = \theta_1/2$, $\phi_2 = \theta_2/2$, $l_2 = (l_1 + l_6)/(2 \times \sin \phi_2)$, $l_4 = (l_5 + l_1)/(2 \times \sin \phi_1)$, as in Fig. 2(a),
(2) If, $\theta_1 \leq U_c$, $\theta_2 \geq 180°$ $\theta_4 \geq 180°$ then, $\phi_2 - \pi - \theta_1$, $\phi_4 - \pi - \theta_1$, as in Fig. 2(b),
(3) If $\theta_1 \leq U_c$, $\theta_2 \leq U_c$, then link $N_3 = N_4$, as in Fig. 2(c),

where U_c represents the upper bounds for an element inner corner.

The evolution process of a quadrilateral mesh under operators in R is shown in Fig. 1. We can see that in each step, the generated finite element mesh is the combination of the previous mesh and the newly generated element; the updated boundary is the result from removing the newly generated element from the previous domain. This process will continue until the domain is filled with required elements.

2.2. Problems of element extraction method

It is obvious from (2) that the necessary and sufficient condition for the rules in R to make a mesh generation algorithm is that they are close for the state Ω_i of mesh generation. It can be observed from (1) that the operation $\cup$ is close for any finite element mesh S, which means that if s_i is a valid element and S_i is a valid mesh, then S_{i+1} would be a valid mesh. However, the operation/may not always be close for domain boundaries, because even if a boundary component b_i is one of the primitive boundary components in Fig. 3, s_i could lead to a component, in B_{i+1}, that does not belong to the primitive boundary components.

In Fig. 4(a), for example, $P_1P_2P_3P_4$ is a good element, but the angle $P_5P_1P_4$ is much smaller than the lower bound for

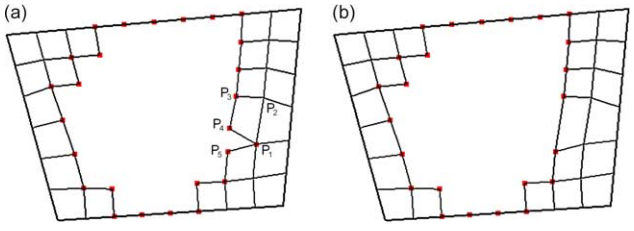

Fig. 4. Boundary updating.

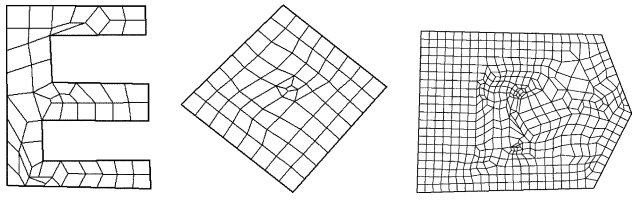

Fig. 5. Bad results from unclosed operations defined by element extraction rules.

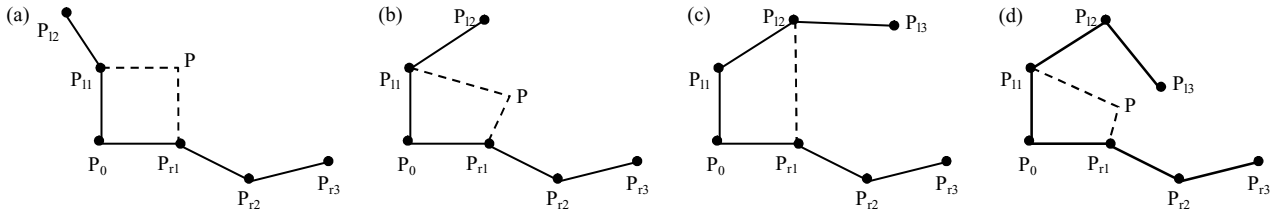

Fig. 6. Challenges in defining element extraction rules.

an inner corner of quadrilateral element. As a result, the boundary component $P_5P_1P_4$ does not belong to the primitive boundary components. In contrast, Fig. 4(b) is a result of the closed operation. Fig. 5 shows some bad results from the operations in R that are not closed.

By comparing Fig. 4(a) and (b), it can be found that the reason of making an operation r_j not closed is that the quality of the element created by r_j is too good to keep the remaining boundary within the range defined by the primitive boundary components. To solve this problem, a set of balanced element extraction rules must be acquired that can generate both a good quality element and remaining boundary components. This is a challenging task due to the complex combination of various boundary components. Fig. 6 lists such an example. In the figure, different locations of boundary points intrigue different rules defined in Fig. 2. Fig. 6(a), (b), and (d) apply rule #2 whereas (c) applies rule #3. The determination of which rule to apply and how to locate point P in the cases of (a), (b), and (d) rely on the relationships between given boundary components.

The acquisition of these element extraction rules is usually done through an interactive trial-and-error process. The success of this interactive process largely depends on the insight and luck of the algorithm developer. The difficulty of this approach is reinforced as the complexity of the problem rises. An automatic rule acquisition method would be essential and even indispensable for the element extraction method to be applied to a wider range of problems. Artificial Neural Network is such a method.

3. ANN-based element extraction: algorithm and system architecture

This section introduces an ANN-based method to acquire the element extraction rules. Two-dimensional quadrilateral mesh generation will be used as an example. The same concept can be applied to more complex problems.

3.1. Description of the algorithm

The procedures of the ANN-based mesh generation are outlined in Fig. 7. An ANN model is trained to identify the relationship between input and output using initial training samples. Training of the ANN model is referred as the calculation of the weight matrix and the bias terms between neurons in different layers. In our study, Back-Propagation (BP) learning algorithm is adopted for the training (Khanna, 1990). Details on the definition of collecting training samples will be elaborated in Section 4. To generate a finite element mesh, a point is chosen from the existing boundary points. This point is called a reference point, around which the new element is generated. The selection of a reference point will be introduced in Section 3.2. Around this reference point, a number of points (called leading points) that have more influence than others on generating a new element are collected from the existing boundary. The input data including the coordinates of the reference point and its corresponding leading points are then

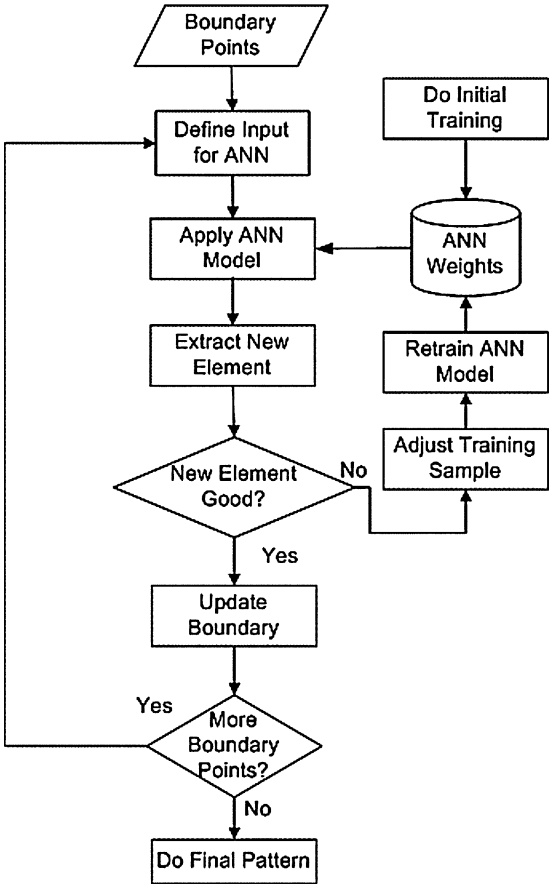

Fig. 7. Flow chart of ANN-based mesh generation.

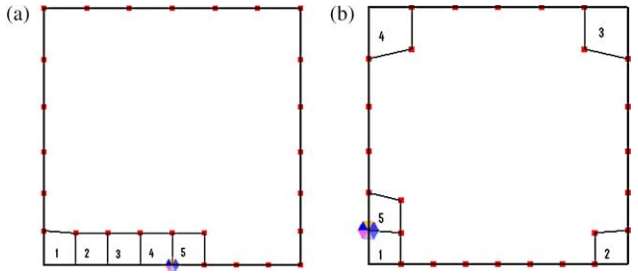

Fig. 8. Finding reference point: (a) newer boundary first; (b) older boundary first.

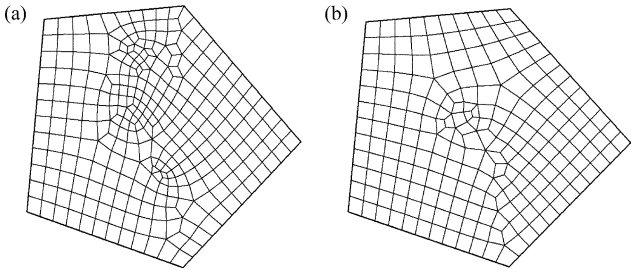

Fig. 9. Mesh generation using different methods to find reference point: (a) newer boundary first; (b) older boundary first.

fed into the initially trained ANN model. The output of the ANN model is the parameters that can represent the new element to be extracted. According to the output parameters, a new element is extracted around the reference point. If the new element is not good enough, the training samples will be adjusted and the ANN model will be retrained until a good quality element is extracted. Then the boundary will be updated for generating the next element. This procedure is repeated until the final pattern is satisfied on the existing boundary. We will have some special considerations for the final pattern, which will be explained in Section 3.4.

3.2. Selection of reference point

A reference point is where we start to extract an element, but not every point on the existing boundary can be taken as a reference point. In our study, if the angle at a boundary point is between 45 and 135°, it may be taken as a reference point. Once a new element is generated, the next reference point needs to be selected. Various criteria can be used to identify the next reference point. Fig. 8 shows two examples. The method in Fig. 8(a) always chooses the first valid reference point next to the current one from the updated boundary. In another method, the updated boundary will not be processed until all the valid reference points in the original boundary have been processed. This is shown in Fig. 8(b). In the figure, the first four elements are generated from the original boundary and element five is generated in the new boundary since no more points meet the requirements of a reference point on the original boundary. If no reference point has been found, then a square element will be extracted around the shortest boundary segment.

Using the two different methods to find reference points for the same domain, different meshes can be generated as shown in Fig. 9. From the figure, it can be seen that the second method can generate more good-quality elements than the first one. The elements close to the periphery have no obvious difference, but there are more bad elements in the center using the first method. In the first method, the local boundary will become bad with the appearance of a bad element and continue to generate bad elements. However, the second method considers more boundary information by choosing all possible reference points from the original boundary and updates the original boundary with all newly generated elements.

3.3. Structure of the ANN for element extraction

It can be seen from Figs. 2 and 6 that the input for element extraction is a set of boundary components and the output is the new element to be extracted. A general input–output relation is shown in Fig. 10. In the figure, a new element is extracted around a reference point P_0. To generate an element around a reference point, all the geometric features that constrain this element should be taken into account as the input. The ideal situation is to take

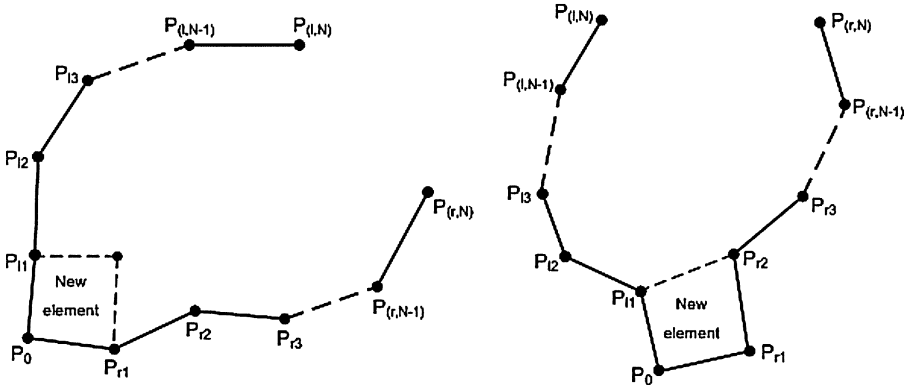

Fig. 10. Leading points.

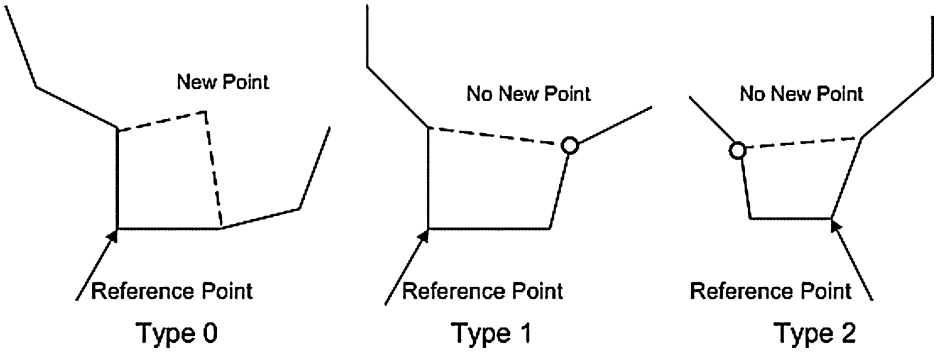

Fig. 11. Three types of new elements.

all existing boundary points and their mutual relationship as the input. However, this is computationally expensive and it is practically impossible to define generic element extraction rules. Too many points in the input of an ANN model means that a domain with the less number of points than that of the input points in the ANN will have to be processed with other methods. This situation may be further complicated by the complex relationships behind large number of points. A trade-off is to take into account a certain number of leading points from the existing boundary that play a significant role in generating an element around the reference point.

In Fig. 10, $2N$ points are picked up around a reference point as the leading points. The number of leading points is selected based on the complexity of the domain. Theoretically, the more complicated is the domain, the more leading points will be included for the input data. More details will be given in Section 5.2. The coordinates of the leading points together with the reference point constitute the input of the neural network model as $[P_{(l,N)}, P_{(l,N-1)}, \cdots, P_{l2}, P_{l1}, P_0, P_{r1}, P_{r2}, \cdots, P_{(r,N-1)}, P_{(r,N)}]$. In this paper, we choose four leading points and one reference point as the input data denoted by $[P_{l2}, P_{l1}, P_0, P_{r1}, P_{r2}]$. The selection and influence of the number of leading points will be elaborated in Section 5.2.

The output includes the parameters to define a new element. In our study, we consider three types of extractions as shown in Fig. 11. Type 0 represents the case where an element is extracted by adding a new point on the base of three boundary points. Types 1 and 2 represent the case where an element is extracted without adding any new point. The difference between type 1 and type 2 is the relative location of the reference point. Type 1 has the reference point on the left corner of the element whereas Type 2 has its reference point on the right corner of the element. We use $[\text{type}, P_n]$ to represent a new element. The variable 'type' should take the values of 0, 1, and 2. P_n is the newly generated point in the new element for Type 0. There is no new point generated for Type 1 and 2, but we choose to include the opposite point of the reference point as P_n circled in Fig. 11 to keep the components of all the output consistent. For the convenience of latter description, P_n is called output point.

Based on the discussions above, the architecture of the ANN for element extraction is shown in Fig. 12. It is indeed a transformation as follows:

$$[\text{type}, P_n] = f([P_{(l,N)}, P_{(l,N-1)}, \cdots, P_{l2}, P_{l1}, P_0, P_{r1}, P_{r2}, \cdots, P_{(r,N-1)}, P_{(r,N)}]) \tag{3}$$

3.4. Final pattern

A final pattern is needed when the number of points left on the boundary is less than the number of input points in the neural network. If five boundary points are used to extract a new element, then the last boundary will have four points left, which form a quadrilateral element. Thus no final pattern is needed. For seven points taken as the input, however, there are only six points left on the last boundary. These six points have to be dealt with separately. For this final pattern, three situations will be considered to complete the mesh generation as shown in Fig. 13 (Zeng & Cheng, 1993).

For the pattern shown in Fig. 13(a), the coordinates of the ith node is

$$\begin{cases} x_i = \dfrac{1}{3}(x_2 + x_4 + x_6), \\ y_i = \dfrac{1}{3}(y_2 + y_4 + y_6). \end{cases} \tag{4}$$

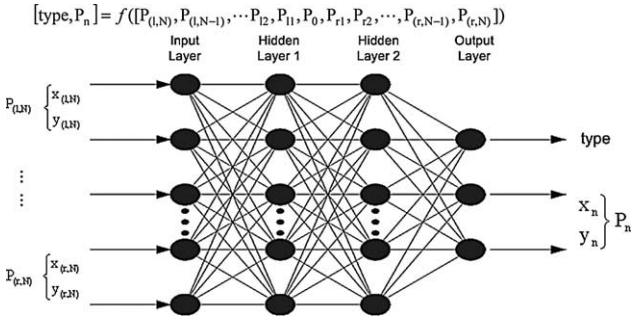

Fig. 12. ANN model structure.

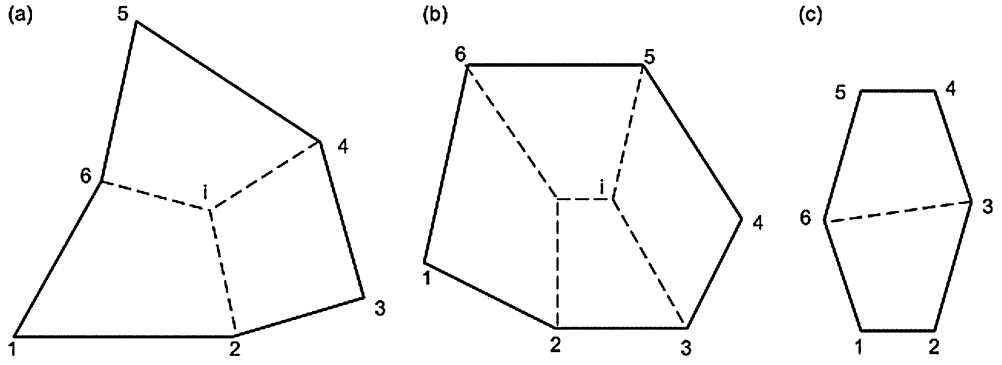

Fig. 13. Final patterns.

For the pattern shown in Fig. 13(b), the coordinates of the ith node is

$$\begin{cases} x_i = \frac{1}{8}(3 * x_3 + 3 * x_5 + x_2 + x_6), \\ y_i = \frac{1}{8}(3 * y_3 + 3 * y_5 + y_2 + y_6). \end{cases} \quad (5)$$

For the pattern shown in Fig. 13(c), connect point 3 and point 6.

It should be noted that if we use nine points or more as the input to extract a new element, the final pattern becomes more complex since more different situations have to be taken into account to complete the mesh generation.

4. Definition and collection of training samples

A key issue in applying artificial neural network is the definition and collection of training samples. The training samples have to be representative and inclusive so that the training results can be extrapolated to other problems. This is especially critical for mesh generation. Unlike general classification problem, ANN-based element extraction method will repeatedly apply the same neural network, by which the original problem is redefined every time. A badly trained neural network may soon generate a boundary that is out of the scope defined by the primitive boundary components, such as those in Fig. 3.

4.1. Generation of training samples

Once the structure of ANN model is determined, a set of training samples can be used to train the ANN model. Theoretically, the training samples should include all possible patterns underlying different combinations of input and output. However, it will be tedious and

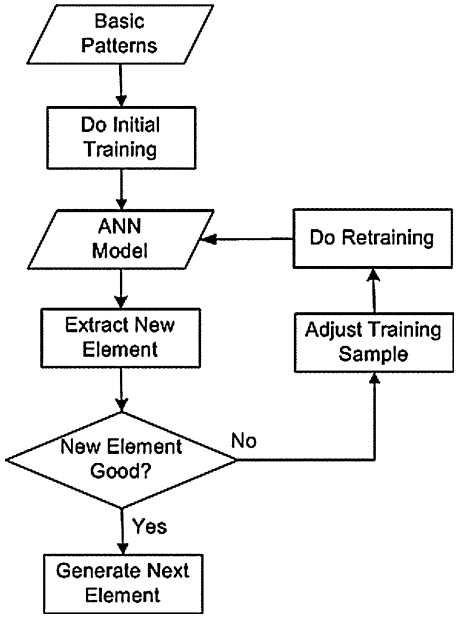

Fig. 14. Procedure of retraining of ANN model.

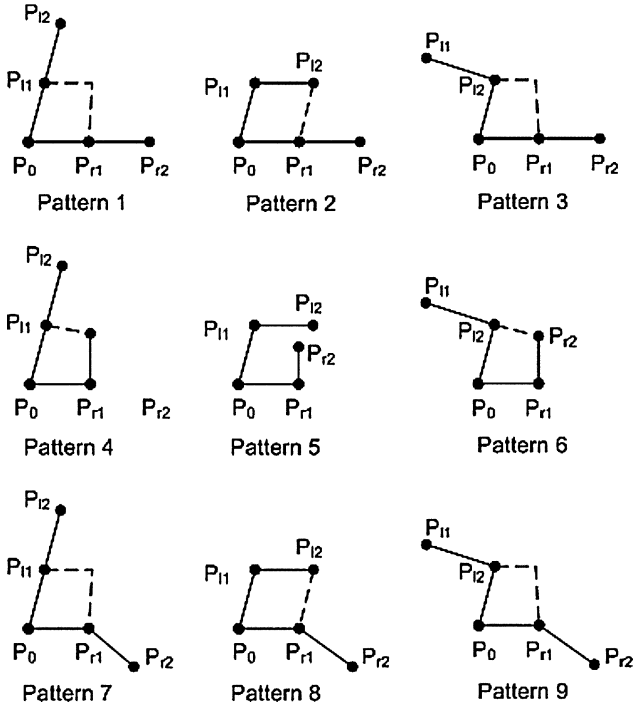

Fig. 15. Nine patterns of input–output relationship.

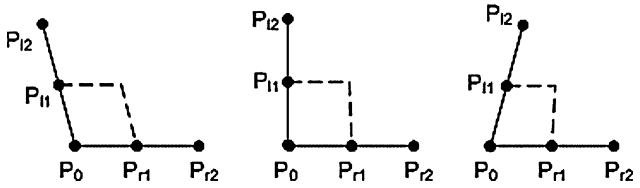

Fig. 16. Different forms of pattern 1.

time-consuming to generate all the patterns. In this paper, we propose a self-learning method that adjusts the training samples by adding new patterns into the initial training samples. First we use a set of training samples that represent the basic patterns to train the ANN model. If the ANN model with the initial training samples fails to generate a good element that belongs to another pattern, then we will adjust the training samples by adding samples of another pattern. The retrained ANN model will be used to generate elements. As shown in Fig. 14, the procedure will be repeated until a good element is generated.

4.1.1. Initial training samples

The initial training samples should combine the basic relationship of input and output. In this paper, we ignore the influence of the relative length of boundary components and consider only three levels (small, medium and large) of angles. The angle of small level is between 0 and 90°; the angle of medium level is between 90 and 180°; and the angle of large level is between 180 and 270°. Therefore, there are $3^{(n-2)}$ different patterns given n leading points. In our study, we used four leading points. As a result, there are $3^2=9$ patterns as shown in Fig. 15. Obviously, pattern 5 is not a valid one, so no initial training samples will be generated for this pattern. For each of other patterns, we used a different angle at the reference point to generate a number of training samples. For example, Fig. 16 shows the different forms of pattern 1. It should be noted that the number of training samples for each pattern is the same. With equal number of samples, each pattern will play an equal role in the training of the ANN model.

4.1.2. Collection of training samples

To generate a training sample from an existing mesh, a set of points including leading points, reference point and the output point can be manually as well as automatically picked up. A user interface has been designed to pick up training samples. Fig. 17 shows the three types of training samples we may collect from a mesh. Five points P_{l2}, P_{l1}, P_0, P_{r1}, and P_{r2} are collected from left to right and the output point is chosen as P_n. It should be noted that the new point P_n is P_{r2} in Fig. 17(b) and the new point P_n is P_{l2} in Fig. 17(c). The coordinates of the six points and the type of the new element will be taken into a training sample. Any mesh with good quality elements may be used for the collection of training samples. Table 1 gives an example of 10 training samples we have collected.

4.2. Normalization of input and output

It can be found from Table 1 that the coordinates of points for input and output data could be any values. However, what matters between input and output are the relative positions of all points in each sample. Furthermore, it is this relative position that can be applied independent of the problems from which the samples were collected. To capture the generic relationship between these points, it is essential to transform all the coordinates into a uniform coordinate system without distorting the relations among the input and output. In the two-dimensional quadrilateral mesh generation problem, the reference point is taken as the origin of the new coordinate system. The vector from the reference point to the first leading point on the right side of the reference point is set as a unit vector along the positive direction of x axis. All the points in each sample are then transformed into this new coordinate system by translation, scaling and rotation. This is shown in Fig. 18.

Given the reference point (x_0, y_0) and the first leading point on the right, (x_{r1}, y_{r1}), we will have the new coordinate system $X'''O'''Y'''$ as is shown in Fig. 18. All the points including leading points, reference point, and the output point in the new element in each sample will be transformed into the new coordinate system $X'''O'''Y'''$.

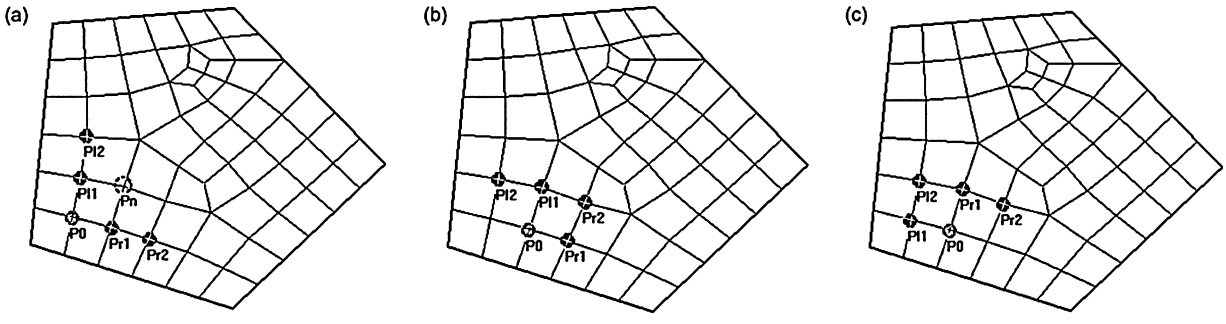

Fig. 17. Three types of patterns collected for training samples: (a) type 0; (b) type 1; (c) type 2.

Table 1
Examples of training sample

Input										Output		
x_{r2}	y_{r2}	x_{r1}	y_{r1}	x_0	y_0	x_{r1}	y_{r1}	x_{r2}	y_{r2}	Type	x_n	y_n
−1.19747	−1.71510	−1.10663	−2.18847	−1.01492	−2.66143	−0.56218	−2.54597	−0.11065	−2.43125	0	−0.65359	−2.08499
7.09138	3.45006	7.19690	2.97116	7.30200	2.49023	7.75940	2.58950	8.21746	2.68208	0	7.65880	3.06881
−1.31411	−22.26263	−1.18250	−22.75125	−1.05347	−23.23812	−0.54303	−23.13958	−0.01878	−22.99969	0	−0.69559	−22.65612
28.87224	−16.62693	28.92887	−17.12458	28.97799	−17.59023	29.24159	−18.02599	29.56558	−17.63935	0	29.35953	−17.43713
−0.83000	−1.15982	−0.74299	−1.62316	−0.65359	−2.08499	−0.20275	−1.98309	−0.29120	−1.53371	1	−0.2120	−1.53371
90.36793	−77.08303	91.75735	−77.30490	91.40188	−78.55770	92.68953	−78.78819	93.05584	−77.56182	1	93.05584	−77.56182
8.12956	−1.27602	8.54664	−1.07961	8.68724	−1.51258	9.11806	−1.28582	8.95484	−0.87355	1	8.95484	−0.87355
−0.36940	−93.02264	−0.16822	−93.50457	0.32499	−93.32099	0.11808	−92.83960	−0.07805	−92.36122	2	−0.36940	−93.02264
28.24149	−15.77552	28.31361	−16.25444	28.80012	−16.14801	28.72781	−15.66914	29.21343	−15.56651	2	28.24149	−15.77552
10.78457	1.51536	10.87277	0.94707	11.40918	1.11712	11.35253	1.69310	11.93000	1.78592	2	10.78457	1.51536

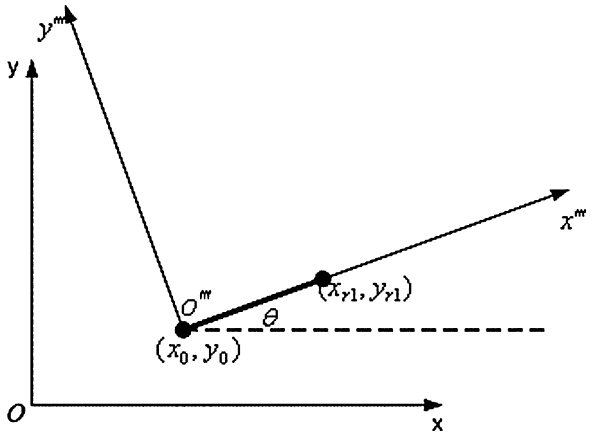

Fig. 18. Coordinate transformation.

A. Translation

$$\begin{pmatrix} x' \\ y' \end{pmatrix} = \begin{pmatrix} x \\ y \end{pmatrix} + \begin{pmatrix} -x_0 \\ -y_0 \end{pmatrix}. \tag{6}$$

B. Scaling

$$d = \sqrt{(x_0 - x_{r1})^2 + (y_0 - y_{r1})^2},$$
$$\begin{pmatrix} x'' \\ y'' \end{pmatrix} = \begin{pmatrix} 1/d & 0 \\ 0 & 1/d \end{pmatrix} \begin{pmatrix} x' \\ y' \end{pmatrix}. \tag{7}$$

C. Rotation

$$\begin{pmatrix} x''' \\ y''' \end{pmatrix} = \begin{pmatrix} \cos\theta & \sin\theta \\ -\sin\theta & \cos\theta \end{pmatrix} \begin{pmatrix} x'' \\ y'' \end{pmatrix}. \tag{8}$$

Using homogeneous coordinates, this transformation can be represented as

$$\begin{Bmatrix} x''' \\ y''' \\ 1 \end{Bmatrix} = \begin{bmatrix} \cos\theta & \sin\theta & 0 \\ -\sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1/d & 0 & 0 \\ 0 & 1/d & 0 \\ 0 & 0 & 1 \end{bmatrix} \times \begin{bmatrix} 1 & 0 & -x_0 \\ 0 & 1 & -y_0 \\ 0 & 0 & 1 \end{bmatrix} \begin{Bmatrix} x \\ y \\ 1 \end{Bmatrix}. \tag{9}$$

The coordinates of the 10 training samples after the coordinate transformation are shown in Table 2. As can be seen from the table, the coordinates of P_0 and P_{r1} remain the same for all samples. In this way, the coordinates of all the sampling points are defined in the same scale and their relationships are also standardized. The actual input for

Table 2
Examples of training sample after transformation

Input										Output		
x_{l2}'''	y_{l2}'''	x_{l1}'''	y_{l1}'''	x_0'''	y_0'''	x_{r1}'''	y_{r1}'''	x_{r2}'''	y_{r2}'''	type	x_n'''	y_n'''
0.1219	2.0591	0.0599	1.0294	0	0	1	0	1.9971	−0.0009	0	1.0542	1.0044
−0.0048	2.0995	−0.0015	1.0518	0	0	1	0	1.9983	−0.0143	0	1.0072	1.0463
−0.1366	1.9374	−0.0662	0.9666	0	0	1	0	2.0412	0.0731	0	0.8881	0.9687
−1.7259	0.8013	−0.8322	0.3907	0	0	1	0	0.6797	0.9373	0	0.1305	0.7966
0.0690	2.0365	0.0316	1.0172	0	0	1	0	1.0277	0.9905	1	1.0277	0.9905
−0.9767	0.9704	0.0987	0.9906	0	0	1	0	1.1105	0.9722	1	1.1105	0.9722
−0.7873	0.9635	0.1587	0.9215	0	0	1	0	1.0977	0.9055	1	1.0977	0.9055
1.0464	0.9927	0.0498	1.0031	0	0	1	0	1.9866	−0.0166	2	1.0464	0.9927
0.9327	1.0257	−0.0673	1.0261	0	0	1	0	1.0598	−1.0231	2	0.9327	1.0257
0.7904	1.0067	−0.2017	0.9511	0	0	1	0	1.0619	−1.0087	2	0.7904	1.0067

the neural network is also reduced to the coordinates of three points.

5. Results and discussions

5.1. Results analysis

In this study, the neural network model is trained with back-propagation (BP) algorithm. A sigmoid transfer function $f(x)=1/(1+e^{-x})$ is used in the BP algorithm. The input data are four leading points and a reference point. The type of new element and the coordinates of the output point are taken as output data. Presently, there are no general rules to determine the number of hidden layers and the number of neurons on each hidden layer for optimal network architecture although there is some research work contributed in this area. In this research, a method using genetic algorithm and direct encoding is used to evolve an optimal network structure. The results will be reported in a separate paper (Kharma, Yao, & Zhu, 2004). Using this method, we have used the ANN architecture with 10 neurons on the input layer, 3 neurons on the output layer, and two hidden layers with 30 neurons in the first hidden layer and 18 neurons in the second hidden layer. Some numerical examples generated by this ANN method are shown in Fig. 19 to display the mesh generation for different boundaries.

As mentioned earlier, element extraction method has the advantage of guaranteeing the quality of elements close to the domain boundary, but the disadvantage is also serious and obvious. To implement the element extraction method, we need to consider the relationships of angles and segment lines to find the basic rules of creating an element based on local boundary features in a heuristic manner. The rules are used to reflect the relationship between boundary information and the element to be extracted. However, it is difficult to define the rules to generate a good-quality element based on all boundary information.

For the purpose of our study, ANN method attempts to solve the problems that element extraction method cannot deal with. The input of the ANN model is a set of leading points that represent the significant information of the existing boundary. The output is the parameters used to describe an element. Using ANN method, we no longer need to develop basic rules to extract an element from the boundary feature manually. A number of training samples are used to learn the relationship between boundary information and the extracted element. We can decompose a complicated domain by correcting bad elements and adding more patterns into the training samples. In this way,

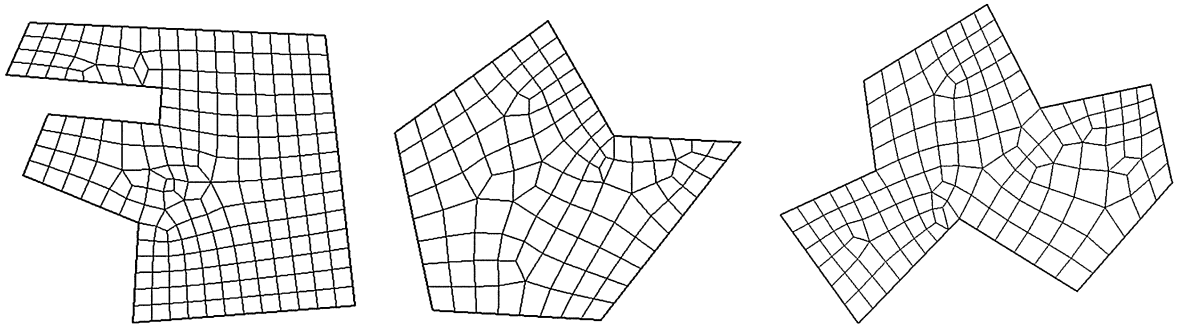

Fig. 19. A few numerical examples generated by ANN method.

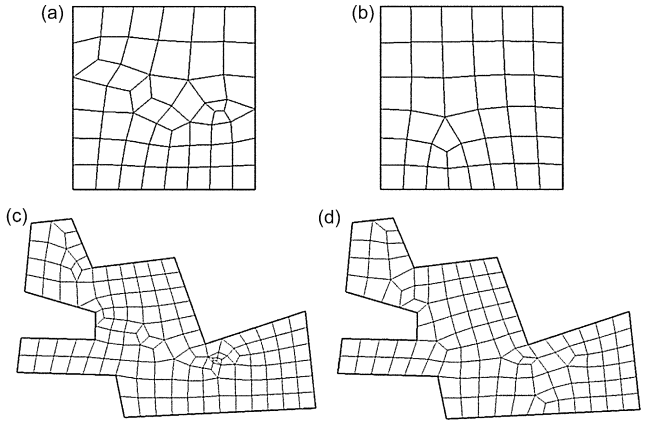

Fig. 20. Mesh generation examples by element extraction method (a), (c) and by ANN method (b), (d).

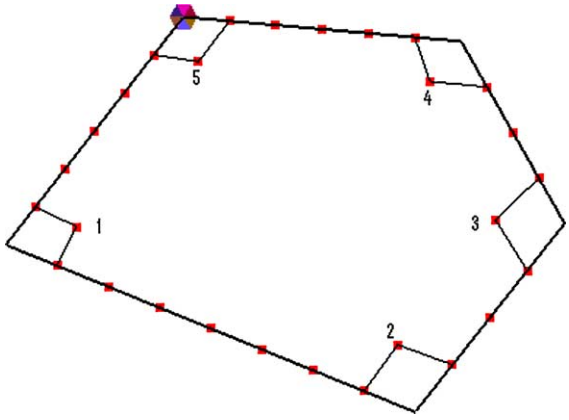

Fig. 21. An example showing the generation of the first five elements.

mesh generation is a self-learning process by adjusting training samples. ANN method provides an alternative method in mesh generation.

Fig. 20 shows a simple example and a complex example generated by the above two methods, respectively. From the results, we can see that the elements around the periphery are still good since the boundary feature is simple. When the boundary feature becomes complex in the center, the basic rules in the element extraction method still consider local boundary information, so they cannot continue to generate good-quality elements. However, with ANN method, we can continue to generate good-quality elements by adding more patterns into the training samples when the boundary condition becomes bad.

In summary, the ANN method has its advantages over rule-based element extraction method in generating elements. First, using training samples, ANN model can learn the relationship between boundary points and the new elements. Secondly, mesh generation is a self-learning process by correcting bad elements and adjusting the training samples. Thirdly, this procedure can be theoretically extended for any mesh generation of 2D or 3D.

5.2. Influence of leading points

As illustrated in Section 3.1, the input of element extraction method includes the reference point and the corresponding leading points. Ideally, all the points on the existing boundary should be taken as the input data for generating a new element. However, too many nodes on the input layer will increase the time spent in generating training samples and training the ANN model. It also

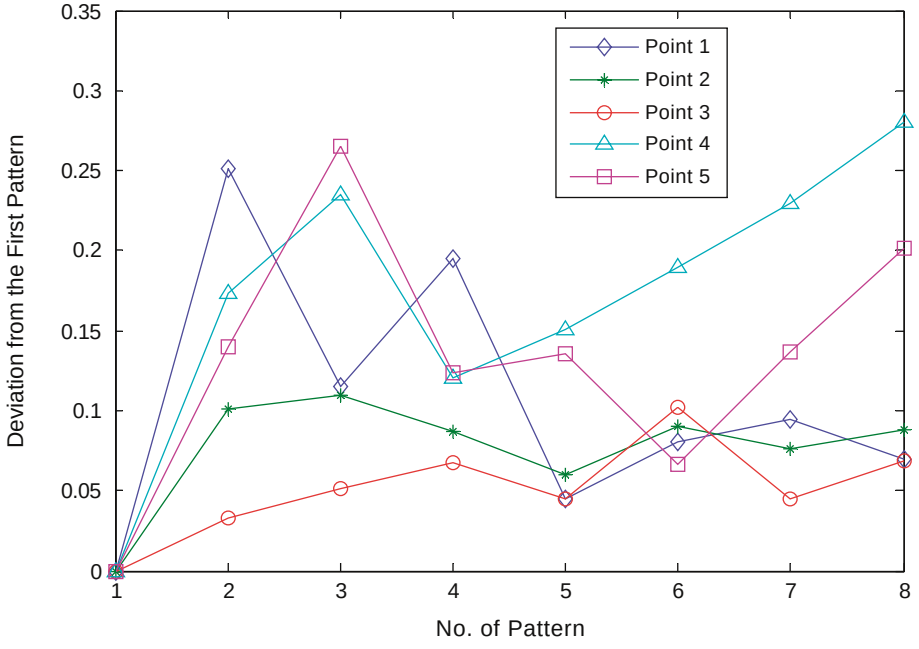

Fig. 22. Deviations of first 5 points from first pattern for 4 leading points.

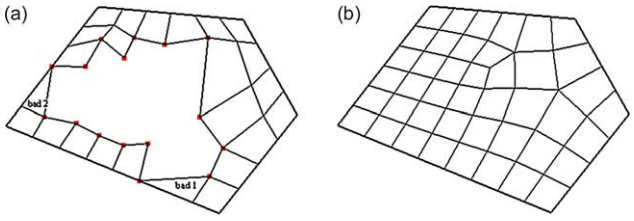

Fig. 23. Mesh generation process of an example.

increases the difficulty in standardizing the input data and sorting out all the situations for the final pattern. We have to make the number of leading points as few as possible. These leading points play a major role in extracting a new element, compared with other points on the existing boundary. In our study, we experimented with four and six leading points to decompose a 2D domain. For four leading points, we used eight basic patterns shown in Fig. 15 to generate a number of initial training samples. It should be noted that different forms of each pattern are included in the training samples and the number of training samples for each pattern is the same in order to make each pattern play an equal role in the training process. However, interactions also exist between patterns. Fig. 21 shows an example of generating the first five new elements. By combining different patterns in the training samples, the deviations of the first five points are changing compared with those generated by the first pattern as shown in Fig. 22. From the figure, we can see that the deviations of the first five points fluctuate with the increase of the number of patterns included in the training samples. So each new element in the final mesh generation is the mutual interactions of all the patterns in the training samples.

Using the initial training samples with these eight patterns, we get two bad elements shown in Fig. 23(a). These bad elements are the results of the interactions of the initial patterns. To correct those bad elements, we need to manually correct the bad element and add the pattern of the corrected element into the training samples. Meanwhile, more samples should be generated and added to the existing samples to enhance the impact of this pattern. Finally, the well-established mesh generation is shown in Fig. 23(b) after the ANN model is retrained.

In the case of six leading points included in the input data, there are $3^4=81$ patterns. Although six leading points can include more information to generate better mesh, the work of generating all training samples for all 81 patterns is huge and tedious. The mutual interactions between these patterns would be greatly complicated. For the same example shown in Fig. 21, Fig. 24 shows the deviations of the first five new points generated by combining 8 patterns in the training samples from those generated by the first pattern. Compared with Fig. 22, Fig. 24 shows smaller deviations. This is because more boundary information has been considered in generating an element. Since eight patterns are only a small portion of all possible patterns, without the enumeration of all 81 patterns in the training samples, the final mesh generation is shown in Fig. 25. The training samples for eight patterns do not include enough information to train the ANN model, which will generate poor elements in the situations that are not covered in

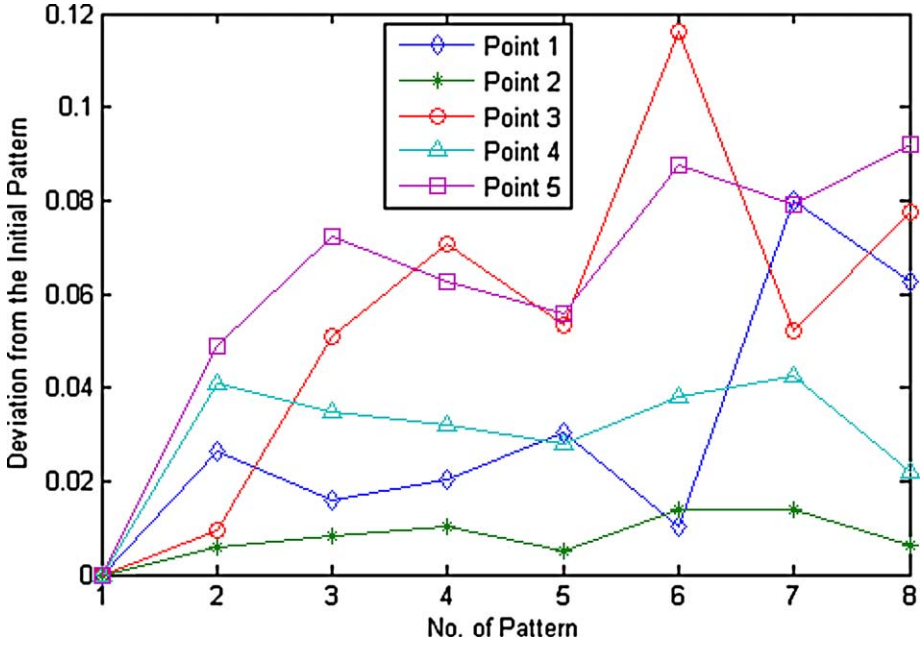

Fig. 24. Deviations of first 5 points from first pattern for 6 leading points.

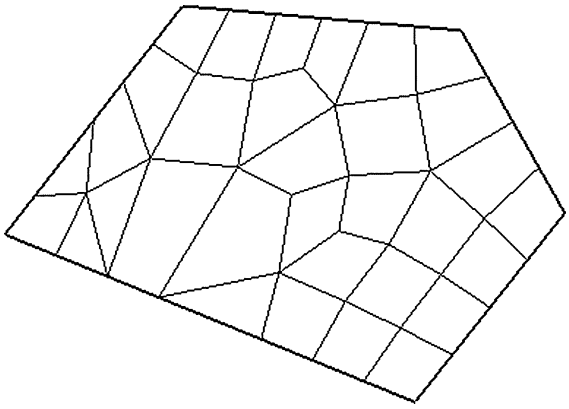

Fig. 25. Mesh generation of the example by 6 leading points.

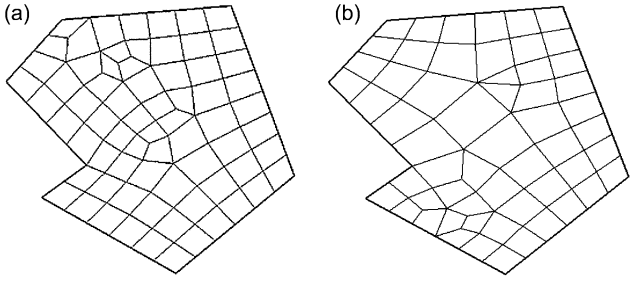

Fig. 26. An example of mesh generation using different number of input points: (a) Five input points; (b) Seven input points.

the training samples. Fig. 26(a) and (b) show another example of mesh generation using four leading points and six leading points, respectively. Obviously the mesh generation in Fig. 26(a) is more regularly and more evenly distributed than that in Fig. 26(b).

From the results, we find that there are a small number of patterns for four leading points and it is easy to include all the patterns in the training sample. In contrary, the number of patterns for six leading points is too big to collect enough training samples and to guarantee good training results for the ANN model. Therefore, five boundary points (four leading points and one reference point) are used as the input data in our final system. If we need to use more leading points to decompose a more complicated domain, all the patterns or some representative patterns for these leading points have to be found out to generate a number of training samples. This work will be explored in the future.

6. Conclusions

In this paper, the BP neural network is used to evolve rules of element extraction for finite element mesh generation. A challenge of this problem is that the problem itself changes in terms of its solution in each iteration. The neural network needs to be able to predict its own performance to avoid the generation of an updated problem that it cannot solve correctly. Therefore, the focus of our discussion is the design of the BP network, including its input, output, as well as its normalization. Based on a coordinate transformation, all training samples from various sources of meshes can be processed in a generic BP network. Experiments have also been conducted to test the influence of the number of points included in the input on the quality of mesh.

It is also found in this paper that the patterns describing the input–output relationship is critical for the success of training the ANN model. For four leading points, we considered all possible patterns and used a user-friendly interface to pick up the corresponding training samples for each pattern to form the initial training samples. We also discussed the influence of six leading points on the training of ANN model. Then a retraining of ANN model is introduced when the initial training samples cannot make the ANN model trained well. Finally the numerical results show that the ANN method is effective in decomposing any 2D domain.

However, we also need to improve the ANN-based mesh generation method in some aspects. First, each training sample is generated by manually picking up six points from an already decomposed domain. It takes a lot of time and energy if we need to generate training samples for many patterns. So it is essential to find a method to automatically generate samples according to each different pattern. Secondly, in our study, BP algorithm is used to train the ANN model. But BP algorithm has some disadvantages in its slow learning rate and its tendency to converge to local minima, as well as its time-consuming when the size of the network architecture is large or the number of training samples is large. Considering these disadvantages of BP algorithm, we will use a better learning algorithm to train the ANN model. These problems will be studied in our future work.

Acknowledgements

This work is partially supported by NSERC (Grant number RGPIN 298255).

References

Ahmet, Ç., & Ahmet, A. (2002). Neural Networks Based Mesh Generation Method in 2-D. *Lecture Notes in Computer Science*, *2510*, 395–401.

Almeida, R. C., et al. (2000). Adaptive finite element computational fluid dynamics using an anisotropic error estimator. *Computer Methods in Applied Mechanics and Engineering*, *182*(3–4), 379–400.

Brown, P. R. (1981). A non-interactive method for the automatic generation of finite element meshes using the Schwarz-Christoeffl transformation. *Computer Methods in Applied Mechanics and Engineering*, *25*(1), 101–126.

Canales, J., Tárrago, J. A., & Hernández, A. (1994). An adaptive mesh refinement procedure for shape optimal design. *Advances in Engineering Software*, *18*(2), 131–145.

Cavalcante Net, J. B., et al. (2001). An algorithm for three-dimensional mesh generation for arbitrary regions with cracks. *Engineering with Computers*, *17*, 75–91.

Cavendish, J. C., Field, D. A., & Frey, W. H. (1985). An approach to automatic three-dimensional finite element mesh generation. *International Journal of Numerical Methods in Engineering*, *21*, 329–347.

Khanna, T. (1990). *Foundations of neural networks*. Reading, MA: Addison-Wesley.

Kharma, N., Yao, S., Zhu, Y. (2005). GA-based Evolution of Multi-hidden Layer Neural Network Architecture (To be submitted to IEEE Congress on Evolutionary Computation, Edinburgh, Scotland).

Lira, W. M., et al. (2002). A modeling methodology for finite element mesh generation of multi-region models with parametric surfaces. *Computers and Graphics*, *26*(6), 907–918.

Manevitz, L., Yousef, M., & Givoli, D. (1997). Finite-Element Mesh Generation Using Self-Organizing Neural Networks. *Microcomputers in Civil Engineering*, *12*(4), 233–250.

Sadek, E. (1980). A scheme for the automatic generation of triangular finite elements. *International Journal for Numerical Methods in Engineering*, *15*, 1813–1822.

Saxena, M., & Perucchio, R. (1990). Element extraction for automatic meshing based on recursive spatial decomposition. *Computers and Structures*, *36*, 513–529.

Shewchuk, J. R. (2002). What is a Good Linear Element? Interpolation, Conditioning, and Quality Measures. In: Proceedings of the 11th International Meshing Roundtable, Sandia National Laboratories.

Topping, B. H. V., Khan, A. I., & Bahreininejad, A. (1997). Parallel training of neural networks for finite element mesh decomposition. *Computers and Structures*, *63*(4), 693–707.

Woo, T. C. (1984). An algorithm for generating solid elements in objects with holes. *Computers and Structures*, *8*(2), 333–342.

Yerry, M. A., & Shephard, M. S. (1984). Automatic three-dimensional mesh generation by the modified octree technique. *International Journal of Numerical Methods in Engineering*, *20*, 1965–1990.

Zeng, Y., & Cheng, G. (1993). Knowledge-based free mesh generation of quadrilateral elements in two-dimensional domains. *Microcomputers in Civil Engineering*, *2*, 259–270.