Research Article

An Automatic Isotropic Triangular Grid Generation Technique Based on an Artificial Neural Network and an Advancing Front Method

1. Introduction

Grid generation is the first step in CFD and other numerical simulations. Grid quality will directly influence the accuracy of simulation results, so grid generation plays an important role in numerical simulations [1–3]. Over the last decades, great progress has been made in grid generation for complex geometries. Generally, there are two categories of grid types, structured and unstructured grids, classified by their geometric topological connectivity. Although structured grids have been widely used due to their excellent computational efficiency and high-order spatial accuracy, grid generation is often tedious and time-consuming for complex three-dimensional (3D) configurations. Conversely, unstructured grids of complex configurations can be generated much more easily than their structured counterpart. Furthermore, compared to structured grids, the unstructured methodology has two obvious advantages. The first advantage is that it is easy to conduct adaptive grid refinement because element splitting and merging can be implemented easily due to the random data structure. Second, unstructured grids have excellent parallelization scalability based on domain decomposition and interface data communication owing to the more balanced load. Therefore, unstructured grids and the corresponding numerical methods have been widely used in engineering applications, and the obvious evidence is that...
most commercial software is developed based on unstructured grids.

After decades of effort, CFD researchers have developed sophisticated unstructured grid generation techniques, such as the AFM [4, 5], Delaunay method [6–8], and quadtree/octree method [9, 10]. These methods have been successfully applied to unstructured grid generation for 2D/3D geometries. Furthermore, some powerful commercial and inhouse grid generation software programs, for example, Pointwise, ICEM-CFD, GridPro, HyperMesh, VGrid, and Capstone, and some open-source codes, such as Gmsh and TetGen, have been developed in the last three decades. The most distinguishing feature of these software programs is their graphic user interfaces (GUIs), which help users handle complex geometries in a visualized and interactive way.

However, as mentioned in NASA’s CFD vision 2030 study [11], grid generation is still a great challenge in modern CFD applications, especially for complex 3D configurations. Since then, a series of special panels in the American Institute of Aeronautics and Astronautics (AIAA) conferences have been held to discuss these issues [12–14]. One of the key issues is how to improve the degree of automation of grid generation because the user interaction operation is very time-consuming and extremely laborious and requires high user expertise. Naturally, some artificial intelligence (AI) methods such as expert systems and neural networks were preliminarily introduced into grid generation techniques to improve the degree of automation. For example, Zeng and Cheng adopted an expert system to solve the grid generation problem in 2D domains; this expert system was used to create rules of mesh generation to replace previous expertise [15]; Yao et al. used an ANN for automatic grid generation [16], and the ANN was used to represent some element extraction rules and to train the relationship behind these rules; Shinobu et al. presented an intelligent local approach for the automatic grid generation of quadrilateral elements, and a fuzzy knowledge processing technology was effectively used to improve the quality of quadrilateral mesh generation [17]; and Guo et al. proposed an optimization method based on ANN to optimize the quality of the triangular mesh and quadrilateral mesh, and this method is more efficient than the traditional optimization-based smoothing methods [18]. However, these earlier works should be continuously promoted, especially in the current time of the rapid development of AI technology. To the best of the authors’ knowledge, there is still no mature application of AI algorithms in high-quality grid generation.

In this paper, an up-to-date machine learning approach based on an ANN is introduced into the traditional AFM of 2D unstructured grid generation. The AFM is a flexible unstructured grid generation technique to easily ensure geometric preservation with complex boundaries. However, this method involves a large amount of geometric computation, such as the selection of active edges, the selection of candidate points, and clumsy intersection checking with the existing fronts. The purpose of introducing an ANN is to improve the efficiency and adaptability of an AFM. The numerical experiments will show that the introduction of an ANN can reduce the complexity of geometric calculation and increase the efficiency by 30% compared with the traditional AFM, and the initial mesh quality generated by ANN_AFM is better than that of the traditional AFM. The specific implementation processes are as follows: First, for illustration, an initial grid over a simple geometric configuration is generated for ANN training by some traditional triangular grid generators, and then an approach for the automatic extraction of the training dataset is proposed. Second, the dataset is input into the ANN to train the network, and then some typical patterns are obtained through learning. Third, after inputting the initial discretized fronts, the advancing front triangular grid generator is launched based on the training results. The system starts from the shortest front, and the adjacent front information is collected as the input of the neural network to choose the best pattern and predict the coordinates of the new point from the trained results during the advancing front process. Finally, after finishing the advancing front process, the initial grid is smoothed to further improve the grid quality. As a preliminary attempt, 2D isotropic triangular grid generation is only considered in this study.

As shown in Table 1, although the time complexity of the two algorithms is the same as that of the traditional AFM, the new method called ANN_AFM can reduce a large number of internal intersection checks; therefore, ANN_AFM has good performance in grid generation efficiency. In complex shape grid generation, an ANN is introduced to improve the adaptability of grid generation. Of course, it will take some time to build the dataset to train the neural network in the early stage of ANN establishment. This paper adopts automatic dataset extraction and preprocessing methods, which greatly saves time for ANN training.

The remainder of this paper is organized as follows. In the next section, the novel isotropic triangular grid generation technique, including the extraction of the training dataset, the preprocessing of the training data, the training of the ANN, the AFM coupled with the ANN, intersection checking, data structure updating, grid size control, and initial grid smoothing, is presented in detail. In Section 3, some typical 2D geometries, including a fighter with a store and a three-element airfoil, are tested to validate the capability of the advancing front triangular generator with an ANN. Finally, some concluding remarks are summarized and future works are discussed in Section 4.

2. Isotropic Triangle Generation Algorithm of the ANN-Based AFM

2.1. Brief Review of the AFM. The AFM generates an unstructured cell by advancing the initial fronts (the discretized boundary) into the computational domain. The basic idea of the AFM is to first divide the boundary of the computational domain into small fronts, such as line segments in 2D and surface triangles in 3D, as the initial front. Then, a newly inserted node or a proper existing point chosen from the active fronts is connected with the target front to construct a new cell (a triangle in 2D and a tetrahedron in 3D). The initial front continues advancing into the computational domain and gradually fills up the entire computational
domain. As shown in Figure 1, all the dotted lines represent the fronts. The segment \( \{P(j), P(k)\} \) is the chosen active front for generating a new cell.

As shown in Figure 2, the AFM procedure can be summarized in the following steps:

1. Input the original discrete geometric boundary, similar to the discrete rectangle in Figure 1. All the discrete edges are called the initial fronts \( F \).

2. When \( F \) is empty, skip to Step 7; otherwise, search for the shortest front \( f \in F \). In practical programming, the heap data structure is used here to store the front information, which can improve the search efficiency.

3. Insert the best point \( P_{\text{best}} \) into the computational domain as a candidate. The position of \( P_{\text{best}} \) is defined as \( P_{\text{best}} = P_m + S_p \cdot \text{normal} \), where \( P_m \) represents the center point of front \( f \), \( S_p \) represents the advancement step at \( P_m \), and \( \text{normal} \) is the unit normal vector pointing to the computational domain.

4. Construct the set of candidate points, \( P_{\text{set}} \) (including \( P_{\text{best}} \)), and sort these points by the grid quality coefficients of the cells to be generated. To choose the points that already existed on the front as much as possible, the quality coefficient, \( Q_{\text{best}} \), of \( P_{\text{best}} \) needs to be discounted (e.g., \( Q_{\text{best}} = 0.6 \times Q_{\text{best}} \)).

5. Pick a candidate point from the top of the set of candidate points, \( P \in P_{\text{set}} \), and connect with the active front \( f \) to form a new possible cell, \( \text{Cell} \).

6. If \( \text{Cell} \) passes through the validity check, such as intersection judgment, then add this cell into the generated grid list, update the active front \( F \), and then return to Step 2; otherwise, return to Step 5 for the next candidate point.

7. Smooth the generated initial grid to improve the quality of the final grid.

There will be a large number of searching and sorting operations for each new cell generation, so the advancing efficiency needs to be further improved.

### Table 1: Comparison of four automatic grid generation methods.

<table>
<thead>
<tr>
<th>Performance/method</th>
<th>Delaunay method</th>
<th>Quadtree/octree method</th>
<th>AFM</th>
<th>ANN_AFM</th>
</tr>
</thead>
<tbody>
<tr>
<td>The computation complexity (( N ) is the total number of grids)</td>
<td>( O(N \log(N)) ) for 2D/3D grid generation.</td>
<td>( O(N \log(N)) ) for 2D/3D grid generation.</td>
<td>( O(N \log(N)) ) for 2D/3D grid generation.</td>
<td>( O(N \log(N)) ); the actual test result is 30% higher than that of the AFM.</td>
</tr>
<tr>
<td>Grid quality</td>
<td>The grid quality is better in 2D, and grid generation may produce thin elements in 3D.</td>
<td>The inner grid quality is good, but the boundary grid quality is poor.</td>
<td>The grid quality is better than those of both the Delaunay method and quadtree/octree method.</td>
<td>The grid quality is better than that of the original AFM.</td>
</tr>
<tr>
<td>Grid density controllability</td>
<td>The grid density can be controlled and local controllability is mediocre.</td>
<td>The grid density can be controlled and local controllability is good.</td>
<td>The grid density can be controlled and local controllability is good.</td>
<td>The grid density can be controlled and local controllability is good.</td>
</tr>
<tr>
<td>Program implementation complexity</td>
<td>Relatively easy</td>
<td>Comparatively complex</td>
<td>Medium difficulty</td>
<td>Medium difficulty</td>
</tr>
<tr>
<td>Anisotropic grid generation</td>
<td>Medium</td>
<td>Difficulty</td>
<td>Excellent</td>
<td>Excellent</td>
</tr>
</tbody>
</table>

![Figure 1: Schematic diagram of grid generation using the AFM.](image)

2.2. ANN-Based AFM Algorithm and System Architecture.

Generally, an AFM is used to generate a grid always from the shortest front, and the inserted new points are determined after intersection checking. However, intersection checking usually consumes many calculations. Through observation and summary analysis of a large amount of data from existing mature grids, the inherent regularity of grid generation may be determined. Therefore, in this work, a novel AFM based on an ANN is developed. A detailed description of this approach is given below.

There are three typical modes for 2D triangular grid generation, as shown in Figure 3. The fundamental goal of the neural network proposed in this work is to identify these three modes and predict the coordinates of the new point in the grid generation process. The following lists the three modes used in this system.

1. If \( \theta_1 \leq \theta_c \) and \( \theta_2 \leq \theta_c \), then generate a new point, \( P_N \), to form triangle \( P_1P_2P_N \), as shown in Figure 3(a), called Mode 1.

2. If \( \theta_1 \leq \theta_c \) and \( \theta_2 \geq \theta_c \), then connect Point \( P_2 \) to \( P_0 \), to form triangle \( P_1P_2P_0 \), as shown in Figure 3(b), called Mode 2.

3. If \( \theta_1 \leq \theta_c \) and \( \theta_2 \geq \theta_c \), then connect Point \( P_1 \) to \( P_3 \), to form triangle \( P_1P_2P_3 \), as shown in Figure 3(c), called Mode 3.

\( \theta_c \) represents the upper bound of an element’s inner corner. In practice, it is set to 60° – 80°.

However, the above modes are only some of the grid generation patterns summarized from a large number of...
standard grids. A large number of “if–else” judgments can make the system more complex. Furthermore, the coordinates of \( P_N \) are affected by the coordinates of the surrounding points. In the classical AFM, a relatively fixed position is directly determined from the normal direction of the midpoint of the front as the coordinate of \( P_N \). After all the grids are generated, the final coordinates of all grid points are obtained by the optimization algorithm. This method results in a poor quality of the initial mesh. Because the ANN has a strong nonlinear expression and generalization ability, it can learn from historical grid data to classify different patterns into corresponding modes and predict the coordinates of \( P_N \), as shown in Figure 3.

Mathematically, this grid generation process can be described by the following equations:

\[
\begin{align*}
    r_k \cdot b_k & \Rightarrow m_k, M_{k+1} = M_k \cup M_k, Q_{k+1} = Q_k - m_k, \\
    \forall b_k & \in Q_k, \quad \exists r_k \in R,
\end{align*}
\]

where \( r_k \) represents one of the modes in \( R \), as shown in Figure 3; \( b_k \) is a part of domain boundary \( Q_k \); and a new grid can be generated according to the mode \( r_k \), thus forming a new active front. \( M_k \) represents the active front generated around \( b_k \); \( M_{k+1} \) denotes the generated grid; and the symbols “\( \cup \) ” and “\( \cdot \)” are set operations, that is, union and difference, respectively.

The flowchart of the ANN-based AFM is shown in Figure 4. The left side is the grid generation section, and the right side is the training data generation section. Our first task is to construct the training dataset and to train the ANN model, which will be discussed in Section 2.3. The grid generation section is similar to the traditional AFM, and the most significant difference between the two approaches is how to generate a new grid. The traditional AFM generates a new cell each time using the front’s normal direction extension and then needs to perform validity checks for all adjacent edges. Therefore, it will take considerable checking time. The new method adopts the well-known ANN for mode recognition. When Mode 2 or Mode 3 in Figure 3 is chosen, the generation process will be much simpler than the previous approach.

2.3. Extraction of the Training Dataset. In this work, the classical AFM is used to generate the initial grids as the machine learning training dataset. The adjacent points of the left and right end nodes of the reference front are taken as the training input. The reference fronts are the grid edges. Theoretically, there are many possibilities for the left and right adjacent points. However, the far edges have little effect on new grid generation. To extract data quickly and conveniently, this work only takes the left and right immediate adjacent points as input. As shown in Figure 5, Points \( P_1 \) and \( P_2 \) are the two end nodes of the chosen front, and \( P_1 \) and \( P_1 \) are the left and right immediate adjacent points of the front, respectively. Different boundary conditions are input into the ANN, and the results are divided into three different modes, as shown in Figure 3.

Generating the training dataset is an important task, so it is necessary to explain the output data format. As shown in Figure 6, in 2D triangular grid generation, a cell is composed of three points. For example, \( \text{Cell}_1 \) contains the three points \([P_1, P_2, P_3]\), while \( \text{Cell}_2 \) contains points \([P_2, P_3, P_4]\). The left cell of \( \text{Face}_2 \) \([P_2, P_3]\), is \( \text{Cell}_2 \); and the right cell is \( \text{Cell}_2 \). With the above data structure, the relationships of points, edges, and cells can be searched easily.

According to the different conditions of the input boundary, the sketch map can be roughly divided into 12 patterns, as shown in Figure 7. Each pattern corresponds to one of the three modes mentioned earlier in the actual training data extraction. Each of the following columns represents a mode in which Mode 1 generates a new point, and Mode 2 and Mode 3 do not.
To obtain the above patterns automatically, an initial grid over a simple domain, as an illustrative example, is used to train the ANN to recognize them, as shown in Figure 8. Then, the Python module, matplotlib, is adopted to restore its physical connection and point label for illustration. Taking the edge \([P_{14}, P_{15}]\) as an example in Figure 8, \(P_{14}\) is the starting point of the edge, and \(P_{15}\) is the endpoint. According to the right-hand system rule, the triangle \([P_{14}, P_{15}, P_{11}]\) can be found, so it can be determined that, regardless of which point is adjacent to \(P_{14}\) or \(P_{15}\), the expected point is \(P_{11}\). For example, the front subset \([P_{10}, P_{14}, P_{15}, P_{10}]\) belongs to Mode 1 mentioned in Figure 3, which will generate a new point \(P_{11}\) and two new edges \([P_{14}, P_{11}]\) and \([P_{15}, P_{11}]\). Of course, the grids are more complex than the above in practical cases. The algorithm for training data extraction is given in Algorithm 1.

2.4. Preprocessing of the Training Dataset. An example of the training dataset of the ANN is shown in Table 2, and the graph corresponding to these data, which are the different cases of one front, is shown in Figure 8. There are many other edges in the grid, so the values of the training dataset are not uniform. In the field of machine learning, different evaluation indicators often have different dimensions and dimension units, which will affect the results of data analysis. To eliminate the dimensional impact among indicators and accelerate the convergence of the ANN, standardization of the training dataset is needed to solve the comparability between data indicators. All indices are on the same order of magnitude after data standardization, which is suitable for comprehensive comparative evaluation. As shown in Figure 8, the starting point \(P_1\) and the endpoint \(P_2\) of the front are used for normalization processing, and the coordinate
values are \((0, 0)\) and \((1, 0)\), respectively, after the final processing.

As shown in Figure 9, the normalization process contains translation, scaling, and rotation \([19]\). The specific translation formula matrix is shown in equation (2), where \((P_{1x}, P_{1y})\) is the coordinate value of \(P_1\) in the \(xoy\) coordinate system.

\[
\begin{pmatrix}
  P'_{x} \\
  P'_{y} \\
  1
\end{pmatrix} =
\begin{pmatrix}
  1 & 0 & -P_{1x} \\
  0 & 1 & -P_{1y} \\
  0 & 0 & 1
\end{pmatrix}
\begin{pmatrix}
  x \\
  y \\
  1
\end{pmatrix}.
\]  

(2)

The coordinate value of \(P_1\) in the \(x'oy'\) coordinate system is \((0, 0)\) through equation (2). Then, the data are normalized by the following equation:

\[
\begin{align*}
\text{len} &= \sqrt{(P_{2x} - P_{1x})^2 + (P_{2y} - P_{1y})^2}, \\
\begin{pmatrix}
  P''_x \\
  P''_y \\
  1
\end{pmatrix} &= \begin{pmatrix}
  0 & 0 & 1 \\
  1 & \text{len} & 0 \\
  0 & 0 & 1
\end{pmatrix}
\begin{pmatrix}
  P'_{x} \\
  P'_{y} \\
  1
\end{pmatrix}.
\end{align*}
\]  

(3)

The coordinate value of the final point can be obtained by multiplying the rotation matrix based on

\[
\begin{align*}
\begin{pmatrix}
  P''_x \\
  P''_y \\
  1
\end{pmatrix} &= \begin{pmatrix}
  \cos \theta & \sin \theta & 0 \\
  -\sin \theta & \cos \theta & 0 \\
  0 & 0 & 1
\end{pmatrix}
\begin{pmatrix}
  P''_x \\
  P''_y \\
  1
\end{pmatrix}.
\end{align*}
\]  

(4)

From the above three equations, (5) can be obtained. This is the normalized coordinate value of any point in the initial coordinate system.
Then, the training data can be obtained after the above transformation, as shown in Table 3, which can be used to train the ANN; and then the appropriate weights and offset values can be obtained by the BP gradient descent method.

The values \([X_{l1}, Y_{l1}; X_{1}, Y_{1}; X_{r1}, Y_{r1}; X_{\text{new}}, Y_{\text{new}}]\) represent the normalized coordinate values of the left adjacent point, the starting point, the endpoint, and the right adjacent point of a typical front, respectively.

For display convenience, the above training data are relatively simple, but the actual initial grid for training data will be more complex, including large concave and/or convex angles. It should be noted that, to achieve a better training effect, it is necessary to reflect the equality of training data; that is, the number of all types of training data is equal and the order is randomly scrambled. In this work, our training data include 15,000 sets of data, 80% of which are used as the training set and 20% are used as the validation and test set.

2.5. Structural Design and Training of ANN. An ANN, composed of many highly connected processing units (or neurons), is an array computing model inspired by biological neural networks and for information processing [20, 21], which is a mathematical model based on the interconnection of the neurons and the relation of the connections (weights and biases) to fitting the majority possible unknown functions. Therefore, an ANN can be regarded as a generalized function to model complex relationships between inputs and outputs or to classify patterns.

A structural diagram of a typical multilayer feedforward neural network architecture is shown in Figure 10(a). The
(1) : Initialize: Import the data file generated by some mature mesh generation software.
(2) : for $i = 1$ to $N$; $N$ is the number of edges.
(3) : Find the vertex of a triangle according to the right-hand’s rule of Edge $i$.
(4) : Find the left and right immediate adjacent points of Edge $i$.
(5) : If: the left adjacent point is the vertex of a triangle, then the mode is 2, as shown in Figure 7.
(6) : Elif: the right adjacent point is the vertex of a triangle, then the mode is 3, as shown in Figure 7.
(7) : Else: the mode is 1, as shown in Figure 7.
(8) : Store data of coordinates with the corresponding mode.
(9) : end for

Algorithm 1: Training data extraction algorithm.

Figure 8: Training data extraction of an ANN.
Table 2: Unnormalized training dataset of a typical edge (or front).

<table>
<thead>
<tr>
<th>Input</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>$X_l$</td>
<td>$Y_l$</td>
</tr>
<tr>
<td>$X_1$</td>
<td>$Y_1$</td>
</tr>
<tr>
<td>$X_2$</td>
<td>$Y_2$</td>
</tr>
<tr>
<td>$X_r$</td>
<td>$Y_r$</td>
</tr>
<tr>
<td>$X_{new}$</td>
<td>$Y_{new}$</td>
</tr>
<tr>
<td>Type</td>
<td></td>
</tr>
</tbody>
</table>

<p>| | | | | | | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>3.0902</td>
<td>9.5106</td>
<td>0.5864</td>
<td>4.9156</td>
<td>-2.0834</td>
<td>6.0788</td>
<td>-3.0902</td>
<td>9.5106</td>
<td>-2.4626</td>
<td>0.8065</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>3.0902</td>
<td>9.5106</td>
<td>0.5864</td>
<td>4.9156</td>
<td>-2.0834</td>
<td>6.0788</td>
<td>-8.0902</td>
<td>5.8779</td>
<td>-2.4626</td>
<td>0.8065</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>3.0902</td>
<td>9.5106</td>
<td>0.5864</td>
<td>4.9156</td>
<td>-2.0834</td>
<td>6.0788</td>
<td>0.8065</td>
<td>2.4626</td>
<td>0.8065</td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3.0902</td>
<td>9.5106</td>
<td>0.5864</td>
<td>4.9156</td>
<td>-2.0834</td>
<td>6.0788</td>
<td>3.0902</td>
<td>9.5106</td>
<td>-2.4626</td>
<td>0.8065</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>3.0902</td>
<td>9.5106</td>
<td>0.5864</td>
<td>4.9156</td>
<td>-2.0834</td>
<td>6.0788</td>
<td>8.0902</td>
<td>5.8779</td>
<td>-2.4626</td>
<td>0.8065</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>3.0902</td>
<td>9.5106</td>
<td>0.5864</td>
<td>4.9156</td>
<td>-2.0834</td>
<td>6.0788</td>
<td>2.4626</td>
<td>0.8065</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0.3962</td>
<td>1.7117</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>2.8634</td>
<td>0.8872</td>
<td>0.3962</td>
<td>1.7117</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>0.3962</td>
<td>1.7117</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1.7876</td>
<td>-0.9422</td>
<td>0.3962</td>
<td>1.7117</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>0.3962</td>
<td>1.7117</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>-0.1579</td>
<td>-1.7899</td>
<td>0.3962</td>
<td>1.7117</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>0.3962</td>
<td>1.7117</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>-0.1579</td>
<td>-1.7899</td>
<td>0.3962</td>
<td>1.7117</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>0.3962</td>
<td>1.7117</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>-0.1579</td>
<td>-1.7899</td>
<td>0.3962</td>
<td>1.7117</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>0.3962</td>
<td>1.7117</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>-0.1579</td>
<td>-1.7899</td>
<td>0.3962</td>
<td>1.7117</td>
<td>1</td>
<td></td>
</tr>
</tbody>
</table>

Figure 9: Normalization processing of input training data. (a) Source; (b) translation; (c) scaling; (d) rotation.

Table 3: Normalized training dataset of a typical edge (or front).

<table>
<thead>
<tr>
<th>Input</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>$X_l$</td>
<td>$Y_l$</td>
</tr>
<tr>
<td>$X_1$</td>
<td>$Y_1$</td>
</tr>
<tr>
<td>$X_2$</td>
<td>$Y_2$</td>
</tr>
<tr>
<td>$X_r$</td>
<td>$Y_r$</td>
</tr>
<tr>
<td>$X_{new}$</td>
<td>$Y_{new}$</td>
</tr>
<tr>
<td>Type</td>
<td></td>
</tr>
</tbody>
</table>

<p>| | | | | | | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>-0.1579</td>
<td>-1.7899</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1.7876</td>
<td>-0.9422</td>
<td>0.3962</td>
<td>1.7117</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>-0.1579</td>
<td>-1.7899</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>2.8634</td>
<td>0.8872</td>
<td>0.3962</td>
<td>1.7117</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>-0.1579</td>
<td>-1.7899</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0.3962</td>
<td>1.7117</td>
<td>0.3962</td>
<td>1.7117</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>-1.2383</td>
<td>0.0782</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>-0.1579</td>
<td>-1.7899</td>
<td>0.3962</td>
<td>1.7117</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>-1.2383</td>
<td>0.0782</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1.7876</td>
<td>-0.9422</td>
<td>0.3962</td>
<td>1.7117</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>-1.2383</td>
<td>0.0782</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>2.8634</td>
<td>0.8872</td>
<td>0.3962</td>
<td>1.7117</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>-1.2383</td>
<td>0.0782</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0.3962</td>
<td>1.7117</td>
<td>0.3962</td>
<td>1.7117</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>0.3962</td>
<td>1.7117</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>2.8634</td>
<td>0.8872</td>
<td>0.3962</td>
<td>1.7117</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>0.3962</td>
<td>1.7117</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1.7876</td>
<td>-0.9422</td>
<td>0.3962</td>
<td>1.7117</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>0.3962</td>
<td>1.7117</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>-0.1579</td>
<td>-1.7899</td>
<td>0.3962</td>
<td>1.7117</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>0.3962</td>
<td>1.7117</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>-0.1579</td>
<td>-1.7899</td>
<td>0.3962</td>
<td>1.7117</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>0.3962</td>
<td>1.7117</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>2.8634</td>
<td>0.8872</td>
<td>0.3962</td>
<td>1.7117</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>0.3962</td>
<td>1.7117</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0.3962</td>
<td>1.7117</td>
<td>0.3962</td>
<td>1.7117</td>
<td>1</td>
<td></td>
</tr>
</tbody>
</table>

Figure 10: Structural diagram of a feedforward neural network. (a) Structural diagram of a forward neural network. (b) Single neuron signal processing model.
network usually consists of an input layer, some hidden layers, and an output layer. In general, each neuron is connected to other neurons of a previous layer through adaptable synaptic weights and bias. By training neural networks, knowledge is usually stored as a set of connection weights (presumably corresponding to synapse efficacy in biological neural systems); this set of data will minimize the error between the actual output value and the expected value of the neural network. Figure 10(b) shows how information is processed through a single neuron. The neuron receives the information from the neurons in the previous layer and then each branch weight is multiplied and bias is added; the result is processed by the activation function as input to the next layer.

The BP algorithm is used in this work to train the neural network to obtain an optimal set of network parameters, which can not only train multilayer feedforward neural networks but also be used for other types of neural networks. This algorithm is based on the strategy of gradient descent, which adjusts the parameters in the direction of the negative gradient of the target, and the error value and the local gradient of the error are used to update the weight and deviation iteratively.

\[
\Delta w_{hj} = \eta \delta_h(n) y_j(n),
\]

where \(\Delta w_{hj}\) is the connection weight between neuron \(h\) in the hidden layer and neuron \(j\) in the next layer, \(\eta\) is the learning rate, \(\delta_h\) is the local gradient for the error in output neuron \(h\) at iteration \(n\), and \(n\) is the number of neural network training iterations.

In this work, the ANN is adopted to recognize different mesh generation modes and predict the new point coordinates. The realization process of the ANN is as follows:

1. Prepare the dataset, extract the features, and feed them to the ANN as input, which has been fully introduced in Section 2.3.
2. Construct the ANN forward propagation grid structure. Forward propagation is the calculation process of building the model, which can provide the corresponding output for a group of inputs.
3. A large amount of feature data was fed into the ANN, and then the ANN parameters were iteratively optimized by the decreasing gradient algorithm.
4. Use the trained model to predict the new point and mode for grid generation.

Machine learning based on ANNs is mainly divided into two processes: the training process and the use process. The training process is a cyclic iteration from Step 1 to Step 3, and the use process is Step 4. Once the parameter optimization is completed, these parameters are then fixed to achieve specific applications. However, the new dataset is directly fed into an existing mature ANN to train the model in some practical applications, and the network structure needs to be changed according to whether the system can correctly respond to the input training dataset. The optimal structure and parameters of the ANN can be obtained by multiple iterations.

The forward neural network should be built after preparing the dataset. The input of the ANN includes a front and approximately \(N\) leading points, as shown in Figure 5. The value of \(N\) in this work is 1 because too many leading points will increase the consumption of computer resources but have little impact on the specific grid generation. Therefore, the input training data are composed of the coordinate points of \(P_{1} \), \(P_{2} \), \(P_{10} \), and \(P_{11} \). The output of the ANN is the specific mode and the coordinates of the selected point. For the multilayer feedforward network, there is no general method to estimate the number of hidden layer neurons. The selection of the number of hidden layer neurons usually relies on experience or dynamic adjustment in practice. Because the theory has proven that the three-layer feedforward network can approximate any continuous function, the three-layer feedforward network gives a good approximation for estimating the number of hidden layer neurons. Therefore, based on the authors’ experience and experimental test, 30 neurons are used on the first hidden layer of the middle layer, 18 neurons on the second hidden layer, 8 neurons on the input layer, and 3 neurons on the output layer. The neural network diagram is shown in Figure 10(a).

Finally, the data of the output layer need to be processed by the nonlinear activation function to improve the nonlinear fitting ability of the ANN. The sigmoid function is chosen as the nonlinear activation function according to the needs of the system as shown in Figure 11. The mathematical expression can be described by the following equation:

\[
f(z) = \frac{1}{1 + \exp(-z)}.
\]

The learning rate, which represents the update magnitude of each parameter, is an important part of neural network training. If the learning rate is too high, the parameters to be optimized will fluctuate around the minimum value and will not converge. If the learning rate is too low, the parameters will converge slowly. The parameters are updated in the direction of the gradient descent of the loss function during the training process. In this study, the exponential decay learning rate is used to train the neural network iteratively. The learning rate \(\eta\) can be written as

\[
\eta = \text{lrb} \cdot \text{lrd} \cdot \frac{\text{global\_step}}{\text{learning\_batch\_size}},
\]

where \(\text{lrb}\) is the initial value of the learning rate, \(\text{lrd}\) is the decay rate of the learning rate, and \(\text{global\_step}\) is the number of training rounds in the current process; and these are untrainable parameters. \(\text{learning\_batch\_size}\) is the amount of input data of the ANN. In this work, \(\text{lrb} = 0.1\), \(\text{lrd} = 0.99\), and \(\text{learning\_batch\_size} = 10\).

In machine learning, the parameters are changed to minimize the loss function and achieve the optimal solution. The core of the optimization algorithm is how to minimize the loss function, that is, how to update parameters.
Gradient descent is one of the most popular algorithms to perform optimization and by far the most common way to optimize neural networks. Based on the basic gradient descent method, there are several classic upgrade algorithms such as stochastic gradient descent (SGD), momentum, and adaptive moment estimation (Adam) [22]. In this work, these three algorithms are used to train neural networks for grid generation. The results show that the Adam optimization not only has a fast convergence speed but also has the lowest loss value in the steady-state region as shown in Figure 12. Furthermore, [23] also confirms that the Adam optimization has high accuracy and the performance is significantly better compared to SGD and momentum. Therefore, the Adam optimizer is selected as the training algorithm in this work.

On the other hand, the optimization methods train a learning model from data to classify and predict results. In the learning process, data are divided into training, validation, and testing datasets. When no validation data are available, the model should depend excessively on the training data, which can lead to overfitting. Overfitting models are often inaccurate because the trend does not reflect the true state of the data. When a neural network model predicts well on known data (training set) but performs poorly on unknown data (testing set), it can be judged whether there is overfitting. The purpose of training a neural network model is to have good generalization performance for any dataset from the training set in the problem domain, enabling the neural network to predict unknown data. To prevent overfitting of neural network training, data augmentation and parameter regularization are used in this study. Data augmentation techniques mainly include rotation, translation, scaling, and other operations on sample data. Since the input features in this work are unrelated to each other, the L2 regularization method is selected. This regularization method puts weight on each parameter \( w \) of the ANN. In other words, the weight is added to each parameter \( w \) in the loss function, and the model complexity index is introduced to suppress model noise and reduce overfitting. As shown in (9), where \( y \) is the expected value and \( y_{ans} \) is the actual value, the first term of the formula is the mean square value of the difference between the actual value and the expected value. The second term is L2 regularization. \( \alpha \) is a hyperparameter and it is 0.1 in this paper, representing the proportion of parameter \( w \) in the total loss.

\[
\text{loss} = \frac{\sum_{i=1}^{n} (y_{ans} - y)^2}{n} + \alpha \cdot \sqrt{\sum |w_i|^2}. \tag{9}
\]

In the ANN training process, all parameters of the neural network are updated to reduce the loss function to obtain a better neural network model. In this study, the final loss drops to 0.02 after 10k iterations, as shown in Figure 12, which achieves the requirements of the grid generation system. The test set verifies the performance of the neural network, and the accuracy is 0.998 (close to 1). Thus the ANN prediction is very close to the practical results. It can adapt to various external environment changes and has a strong generalization ability.

2.6. Intersection Checking. Intersection checking, which is not only related to the success of grid generation but also directly affects the efficiency of the generation system, is very important to grid generation using an AFM. Experience shows that small changes in intersection checking operations can seriously affect the resulting grid structure. Although the ANN-based machine learning approach is introduced into the AFM, intersection checking should be conducted when a new point is inserted into the computational domain. From the perspective of computational geometry, the human eye can quickly distinguish whether two triangles intersect with each other, but implementation by computer code is a complicated task.

As illustrated in Figure 13, the new fronts [Pa, Pnew] and [Pb, Pnew] added by the ANN will intersect with the other nearby fronts. From the point of view of the algorithm, it is
considered that the two triangles do not intersect as long as their edges do not intersect with each other. The algorithm has been discussed in [24] in detail and is not repeated in words here. In future work, AI image identification technology will be utilized for intersection checking.

2.7. Data Structure Update. As shown in Figure 14, the boundary of the computational domain is first discretized into segments \([P_1, P_2], \ldots, [P_n, P_0], \) and \([P_n, P_1]\); and each segment is called an edge or a front. All fronts can be divided into active fronts and/or inactive fronts; and the active front can generate grid elements, such as \([P_3, P_5]\) and \([P_n, P_1]\). An inactive front cannot regenerate a new grid element, such as \([P_5, P_n]\). It should be noted that the advance process of the front needs to follow the right-hand rule, such as the front \([P_3, P_4]\), which generates a new point \(P_{n+3}\), and the three points \([P_3, P_4, P_{n+3}]\) form a new triangle called \(\text{Cell}_3\). All of the initial boundary segments are active fronts at the beginning of the AFM.

As shown in Figure 3, there are three typical modes for triangular grid generation. A new point \(P_{n+1}\) is inserted in Mode 1; two new active fronts, \([P_{n+1}, P_1]\) and \([P_2, P_{n+1}]\), should be added to the front heap, while the front \([P_1, P_2]\) will be deleted from the front heap and set as an inactive front. On the other hand, there is no new point in Mode 2 and Mode 3; then an active front, for example, \([P_3, P_4, P_{n+3}]\), will be added to form the triangle \([P_3, P_4, P_{n+3}]\), and two inactive fronts, \([P_3, P_4]\) and \([P_{n+3}, P_3]\), need to be deleted from the active front heap.

There are some special cases in the generation process. As shown in Figure 15(a), supposing that the front \([P_1, P_2]\) is the chosen active front to generate a new grid, the data subset fed into the ANN may be \([P_3, P_1, P_2, P_3]\). This type of data subset should be classified as Mode 3 through the ANN, as shown in Figure 3; therefore, the system will connect the edge \([P_1, P_3]\) directly in this mode, while the front data updating is different from before. The new edge will not belong to the active front because both sides have a cell, and the system will delete four inactive fronts. Another case is shown in Figure 15(b). If the active front is \([P_1, P_2]\), the data subset \([P_3, P_1, P_2, P_3]\) will be classified as Mode 1 through the ANN. Two edges will be generated according to the normal situation, but the two edges, \([P_2, P_3]\) and \([P_3, P_1]\), do not belong to the active front. The system will delete these inactive fronts, that is, \([P_1, P_2], [P_2, P_3], [P_3, P_1]\). Therefore, in the actual front advancement process, whether there are already two cells on the left and right sides of the front should be checked. If two cells share a front, then the front should be deleted from the active front heap.

2.8. Grid Size Controlling. In practical numerical applications, the grid distribution should match the problem to be simulated. In other words, the grid size should be controlled in the computational domain. As shown in Figure 16(a), the white cylinder represents the configuration to be considered, and the green part denotes the flow field. When solving the flow field governing equations on the computation domain, it is necessary to distribute a dense grid over the configuration to simulate the flow field more accurately near the solid wall. Furthermore, in the far-field region, relatively coarse grids are more appropriate to improve simulation efficiency since the physical quantities change smoothly; therefore, a smooth grid size distribution from the solid wall to the far-field is needed. Here, the exponential function is adopted to control the advancing step size of the grid, as shown in Figure 16(b).

The relation between the distance to the solid wall and the grid size is fitted with the exponential function of

\[
y = e^{ax} + b,
\]

where \(x\) represents the distance between the new front to be advanced and the nearest surface of the configuration and \(y\) is the actual grid size. As shown in Figure 16(b), when the distance to the solid wall is 0, the step length is set to \(\text{len}(P_1, P_2) \times \sqrt{3}/2\). In the far-field region, the distance to the solid wall is \(R\), and the step length is set to \(\text{len}(P_n, P_{n+1}) \times \sqrt{3}/2\), where \(\text{len}(P_1, P_2)\) represents the distance function of two points.

2.9. Grid Quality Optimization. The long-term CFD practice shows that grid quality is very important to the accuracy of the simulation results. For complex geometries, poor grid quality may lead to divergence of the calculation process, and then the flow solver will fail to break down. Therefore, it
is generally necessary to optimize the initial grids after AFM. The basic requirements of CFD are smoothness, orthogonality, and a reasonable grid distribution [25–27]. There are two main approaches to smooth the grid distribution, that is, diagonal transformation based on the Delaunay criterion and node relaxation.

2.9.1. Diagonal Transformation. In the grid generation process, the inner angles of some triangles may be very small, and these triangles are cells with poor grid quality. As shown in Figure 17, when the diagonal AC of quadrilateral ABCD is replaced by BD, the grid quality will be improved. The Delaunay criterion is used here to judge the grid quality [28, 29]. Figure 18 shows an example before and after the diagonal transformation. It shows that, after the iterations of the diagonal transformation, the triangles with obtuse angles have been eliminated.

2.9.2. Node Relaxation Based on Spring Analogy. After the diagonal transformation, the node relaxation approach is adopted to further improve the grid quality [30, 31]. The essence of this method is to move the position of each internal point in the grid to the center of the polygon around itself to reach an equilibrium state, similar to a spring system; that is,

\[
\begin{align*}
    x_p &= \sum_{i=1}^{N} \frac{x_i}{N}, \\
    y_p &= \sum_{i=1}^{N} \frac{y_i}{N},
\end{align*}
\]  

(11)

where \((x_i, y_i)\) is the coordinate of Node \(i\) and \(N\) is the total number of points connected to Node \(i\). For example, the computational grids before and after spring relaxation are shown in Figure 19. Obviously, the grid quality of Figure 19(b) is better than that of Figure 19(a).

In actual applications, a few smoothing iterations of diagonal transformation and node relaxation may be needed to further improve the grid quality.

3. Grid Generation Examples and Discussions

Based on the details discussed in Section 2, the framework of an ANN-based AFM is developed. The code running environment is as follows: Ubuntu 16.0.4, TensorFlow 1.2.1, Python 2.7.1, and the module “matplotlib” for visualization. In this section, some typical computational domains are adopted to generate high-quality isotropic triangular grids to validate the capability of the present ANN-based AFM.

First, the designed system is used to discretize some regular and irregular polygons with uniform grid sizes.
Figure 20(a) shows the initial grids in an irregular polygon generated by the ANN-based AFM generator; and Figure 20(b) gives the final grids after optimization, which are smoother than before optimization. Similarly, Figure 20(c) shows the initial grids in a pentagram, and the final grids after optimization are shown in Figure 20(d). Once again, a very smooth grid distribution in the pentagram will be obtained.

Next, some practical cases for CFD simulations will be shown to demonstrate the effectiveness of the proposed method. For these cases, the grid size controlling function which is presented in Section 2.8 is adopted to distribute grids reasonably. Figure 21 illustrates the computational domain over an airfoil. As shown in Figure 21(a), the initial discrete boundaries of the computational domain and the airfoil are placed at the center of the far-field circle. To
Figure 20: Triangle generation and optimization of simple configurations. (a) Initial grids in an irregular polygon. (b) Final grids after optimization for the polygon. (c) Initial grids in a pentagram. (d) Final grids after optimization in the pentagram.

Figure 21: Grid generation over an airfoil. (a) Initial front over an airfoil. (b) Final triangular grids over the airfoil.
discretize the airfoil more accurately, more grid points are located in the leading edge and trailing-edge regions. Therefore, the grid sizes in these two regions are relatively small, as shown in Figure 21(b), the final triangular grids over the airfoil. It should be noted that the grids transfer smoothly from the solid wall to the far-field circle.

The next case is a fighter with a released store, which is more complicated than the airfoil above. Figure 22 shows the final grids over the joint configuration. Once again, the grids near the noses and the trailing edges of the fighter and the store are refined. In the regions with large curvatures, such as the inlet and the vertical tail, the grids are also refined to capture the geometry more accurately.

The last case is a three-element airfoil, which is a typical configuration for the taking-off or landing of an aircraft. The final triangular grids are shown in Figure 23. Similar to the fighter case, the grids near the leading edges and the trailing edges of each component of the three-element airfoil are refined. Obviously, this grid distribution is more suitable for CFD simulations.

Then, the quality of the generated grids will be checked. The quality coefficient $q$ of a cell is defined as twice the ratio of the inscribed radius over the circum-radius of the cell. Therefore, obviously, $0 \leq q \leq 1$. The larger the coefficient is, the better the grid quality is. Table 4 lists the grid quality coefficient distributions of some practical cases in Section 3, where Cases 1–3 represent the single airfoil, the fighter with a store, and the three-element airfoil, respectively. Note that the quality coefficients of most of the initial grids (85%) generated by the ANN-based AFM are larger than 0.9 in Case 1. After optimization, the proportion of grids whose quality coefficient is larger than 0.9 is more than 92%, and that of the grids whose quality coefficient is less than 0.6 is less than 0.3%. Therefore, the grids are of much higher quality for CFD simulations.

Finally, for Case 1, the grid quality and consumption time of the ANN-based AFM and the original AFM are compared to demonstrate the potential advantage of the proposed ANN-based AFM. The initial grids before grid smoothing by the two approaches are shown in Figure 24. The grid quality in Figure 24(b) is better than that in Figure 24(a), especially in the boundary junctions and complex areas of the grid. Therefore, the initial mesh quality generated by the traditional AFM and ANN-based AFM is compared; statistics show that the number of cells with grid quality coefficients lower than 0.6 generated by the traditional AFM method is twice that of the ANN-based AFM, which makes subsequent mesh optimization easier, thus further saving optimization time. The main reason why the
proposed ANN-based AFM can improve the quality of the initial grid is that the position of the newly inserted point in Mode 1 is determined by the trained ANN instead of the fixed point in the original AFM (\(P_{\text{best}} = P_{m} + S_{p} \cdot n_{\text{ab}}\) in Section 2.1). This benefits from the good generalization of the neural networks, which is also the performance of learning ability.

Grid generation efficiency is an important aspect, as shown in Table 5; the efficiency of ANN_AFM is about 30 percent higher than that of the traditional AFM. In the grid generation procedure, Figure 24(b) shows that there are 1351, 609, and 469 grids in Mode 1, Mode 2, and Mode 3, respectively. The corresponding CPU times of the three modes are 1.6, 0.8, and 0.8 microseconds. The main reason for the time difference is that Mode 1 contains a large number of intersection checks, which leads to a loss of efficiency, while the other two modes do not. However, for the traditional AFM, each front advancement process is almost

![Figure 23: Triangular grid generation over a three-element airfoil. (a) Global view. (b) Enlarged view over the leading edge. (c) Grids near the three-element airfoil. (d) Enlarged view over the flap.](image)

<table>
<thead>
<tr>
<th>(q)</th>
<th>Initial grid</th>
<th>After optimization</th>
</tr>
</thead>
<tbody>
<tr>
<td>(q)</td>
<td>Case 1</td>
<td>Case 2</td>
</tr>
<tr>
<td>(0.9 \leq q \leq 1)</td>
<td>85.79</td>
<td>75.36</td>
</tr>
<tr>
<td>(0.8 \leq q &lt; 0.9)</td>
<td>8.64</td>
<td>16.04</td>
</tr>
<tr>
<td>(0.7 \leq q &lt; 0.8)</td>
<td>2.85</td>
<td>5.35</td>
</tr>
<tr>
<td>(0.6 \leq q &lt; 0.7)</td>
<td>1.53</td>
<td>1.71</td>
</tr>
<tr>
<td>(Q &lt; 0.6)</td>
<td>1.19</td>
<td>1.52</td>
</tr>
</tbody>
</table>
equivalent to Mode 1, so it will take more CPU time than the ANN-based AFM. Thus, for Case 1, the total CPU time of the ANN-based AFM is approximately half of that of the traditional AFM. If the new method is extended to complex 3D geometries, the grid generation efficiency can be improved by preliminary prediction. That will be the authors’ future work.

4. Concluding remarks and Future Works

This study aims to solve a challenging problem: learning knowledge from existing grid data for achieving high-quality meshes in both the boundary and interior of complex geometric domains.Conventionally, to ensure the overall quality of the grid, the traditional AFM relies heavily on heuristic operation, which is extremely difficult, expensive, and time-consuming for grid algorithm developers. In this study, a novel automatic triangle generation method is proposed for complex 2D geometry. A training data extraction method is developed first to reduce the complexity of manual dataset production and increase the sample integrity of learning. Then, the ANN model with two hidden layers is used to recognize the generation mode of the new grid to help the classical AFM. The performances of different optimization algorithms are compared and the best one is selected for the neural network training. When the initial mesh is generated by the ANN-based AFM, grid optimization techniques are adopted to improve grid quality. In the process of neural network training, the uniformity of the number of samples should be considered. Learning samples that are too simple will lead to the inaccurate classification of some modes by the ANN when applied to complex geometries. Finally, some typical 2D geometries are tested to validate the capability of the ANN-based AFM. The experimental results show that the ANN can correctly classify various complex modes, predict the coordinates of the new points, and finally improve the adaptability of the original AFM. The efficiency of the proposed method is about 30 percent higher than that of the traditional AFM, and the number of poor meshes is reduced, which improves the quality of the initial mesh.

However, the present ANN-based AFM is only a preliminary attempt for 2D cases. It is still essentially a type of AFM. Since only two immediate adjacent fronts are considered for mode recognition by the ANN-based approach, it still needs time-consuming interaction checking in partial mode, even though two other modes generate grids directly without interaction checking, and the initial mesh quality also needs further improvement. In addition, only isotropic triangles are considered in this study, but, for realistic viscous flow simulations, anisotropic grids with high aspect ratios should be generated in boundary layers. Furthermore, triangle/quadrilateral hybrid grids are more suitable for high Reynolds number problems. Therefore, in future works, this approach will be extended to unstructured quadrilateral grid generation, triangle/quadrilateral hybrid generation, high aspect ratio grid generation in boundary layers, and automatic hybrid grid generation over complex 3D geometries. In addition, more effective machine learning, such as deep reinforcement learning, will be introduced into the grid generator in this work; and more parameters or operations, such as the grid distribution in the computational domain, intersection checking with the image recognition technique, the effect of geometric curvatures, and more adjacent fronts...
for mode recognition, will be considered with modern AI approaches. The long-term goal is to develop a fully automatic high-quality grid generation system.

Data Availability

The data that support the findings of this study are available from the corresponding author upon reasonable request.

Conflicts of Interest

The authors declare that they have no conflicts of interest regarding the publication of this paper.

Acknowledgments

This research was funded by the Innovation Foundation of State Key Laboratory of Aerodynamics (Grant no. 190104), the National Key Project (GJXM92579), the Open Research Program of State Key Laboratory of Aerodynamics (20200203), and the Technology Innovation and Application Development Special Project of Chongqing (cstc2020jscx-msxm0434).

References


