A Matrix Manipulation Approach to Graph-Based Feature Extraction in Wooden Beam Machining

Shu-Ting Chang* and Alan C. Lin**

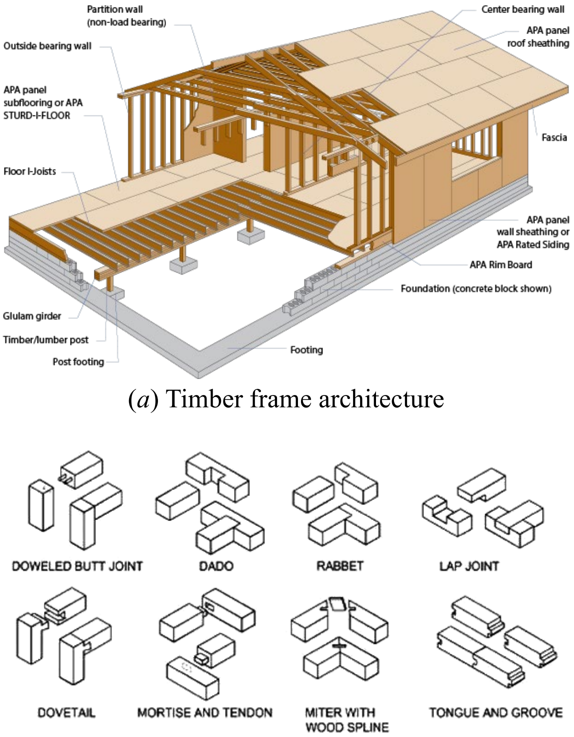
Keywords： CAD/CAM, wooden beam machining, feature recognition, process planning

ABSTRACT

In the woodworking beam processing industry, engineers typically rely on CAM software to generate machining codes, which are then executed by NC machines equipped with automated feeding systems. This paper introduces a matrix manipulation approach for the automated extraction of both standard and intersecting features in wooden beam machining, enabling 3D CAD models to be seamlessly transformed into complete toolpaths. The proposed system analyzes the topological and geometric properties of solid beam models and employs a graph-based method, augmented by matrix permutation and feature encoding, to accurately identify and classify various types of machining features. It further determines suitable machining methods, parameters, and tool orientations, while optimizing the machining sequence to reduce tool changes and travel distances. Implemented in C++ on Siemens NX's development platform, NX Open, the system is validated through case studies on real wooden beam samples with complex features. The results demonstrate its effectiveness, scalability, and strong potential to advance intelligent CAD/CAM integration in wooden beam manufacturing.

INTRODUCTION

Figure 1(*a*) illustrates a lightweight timber frame structure (Zhao et al. 2012). Timber frame construction has increasingly emphasized modularization, dimensional standardization, and automated production, resulting in structural features such as slots, steps, pockets, and holes, as shown in Figure 1(*b*) (Cao 2010). These features can be efficiently machined using multi-axis NC machining centers that integrate sawing, milling, slotting, and drilling, thereby reducing labor intensity and improving accuracy and productivity. To support such production, computer-aided process planning (CAPP) plays a pivotal role in automatically translating CAD models into executable toolpaths.



(*a*) Timber frame architecture

(*b*) Common wooden beam connections

Fig. 1. Lightweight timber

This study builds on the core concept of CAPP, aiming to automatically convert solid beam models into machining features and toolpaths. In toolpath planning, several studies (Liu et al. 2017; Manafi et al. 2017; Watanabe and Nakamoto 2020) have addressed feature-specific toolpath generation. For feature recognition, Han (1998) and Shah et al. (2000)

Paper Received September, 2025. Revised November, 2025, Accepted December, 2025, Author for Correspondence: Alan C. Lin.

* *M.S. Student, Department of Mechanical Engineering, National Taiwan University of Science and Technology, Taipei 106, ROC.*

** *Professor, Department of Mechanical Engineering, National Taiwan University of Science and Technology, Taipei 106, ROC.*

reviewed 25 years of research, identifying five major methods: volumetric decomposition, rule-based, hint-based, graph-based, and AI-based approaches. Volumetric decomposition defines features by aggregating material removal volumes based on predefined rules and includes two categories: convex-hull (Han et al. 2000; Li et al. 2002) and cell-based decomposition (Ding et al. 2022). Rule-based methods apply If-Then-Else logic to identify features that meet specific conditions (Babić et al. 2008). Hint-based methods extend this logic by identifying ambiguous features using predefined hints (Jones et al. 2006; Verma and Rajotia 2008; Verma and Rajotia 2010).

The graph-based method, most widely adopted in feature recognition, represents topological and geometric relationships through graphs, with nodes and edges denoting surfaces and boundaries. It uses attributes such as edge convexity, surface type, perpendicularity, and tangency, and recognizes features by matching subgraphs to predefined patterns (Lambourne et al. 2021; Colligan et al. 2022; Ning et al. 2023; Wu et al. 2024; Yang et al. 2025).

In recent years, artificial intelligence (AI) techniques, most notably deep learning and neural networks, have gained traction in feature recognition. Shi et al. (2020-1) reviewed AI applications in this area, noting that model effectiveness depends on the selection of input vectors and network architecture. Lambourne et al. (2021) and Babić et al. (2011) developed AI models using different mathematical and geometric inputs. Other studies applied AI to various machining feature types (Shi 2020-2). For example, Shi et al. (2020-3) proposed a method to calculate residual heat across 3D surfaces over time, while Marchetta and Forradellas (2010) extended AI use to both feature recognition and process planning task sequencing. Colligan et al. (2022) developed a hierarchical deep learning model (Hierarchical CADNet) that directly learns from B-Rep structures for feature recognition, achieving robust performance on complex machining models. Ning et al. (2023) further demonstrated the feasibility of applying convolutional neural networks to machining feature recognition, reporting substantial improvements in recognition rates for intricate geometries. Building on these advances, Wu et al. (2024) introduced AAGNet, a graph neural network framework that leverages attributed adjacency graphs to handle multi-task feature recognition, significantly reducing errors in feature interaction scenarios. Meanwhile, Khan et al. (2024) extended automatic feature recognition to hybrid additive-subtractive manufacturing, integrating dimensional attribute extraction into the workflow to enable intelligent process planning. Most recently, Yang (2025) proposed a hybrid recognition framework that combines primitive decomposition with learning-based reconstruction to accurately detect highly interacting machining features, underscoring the growing importance of hybrid AI–graph-based solutions.

This paper presents the development of a CAPP system for slender wooden beams with numerous geometric features. To resolve ambiguity in graph-based methods, where a single graph may correspond to multiple features, a matrix permutation and feature encoding approach is proposed, ensuring a one-to-one mapping between graphs and features. This enables fully automated recognition of both standard and intersecting features. Even with a large number of features and surfaces, the system accurately identifies all features and rapidly generates process data.

The research addresses four key issues:

(1) Extraction of topological and geometric information: The 3D CAD model of the beam is analyzed to classify its topology and geometry. A face connectivity graph is constructed based on face adjacency and edge attributes to isolate each machining feature.

(2) Feature recognition: A face connectivity matrix is derived from each feature's graph, reordered to generate a feature code, and matched against a feature database to determine the feature type.

(3) Process planning and sequence arrangement: For each recognized feature, appropriate tools, machining methods, and parameters are assigned. The machining sequence is arranged to balance precision and efficiency.

(4) Toolpath generation: Toolpaths are generated in sequence according to the planned operations. After verification, a post-processed NC program is produced.

Detailed methodologies for these four components are presented in the following sections.

AUTOMATED DETECTION OF CONCAVE AND CONVEX FEATURES

Figure 2 illustrates the topological structure of solid geometry, comprising features, surfaces, loops, edges, and vertices. This structure can be represented using a face connectivity graph (FCG), where nodes denote faces and lines represent edges - solid lines for concave edges and dashed lines for convex edges. For instance, the beam in Figure 3(*a*) contains 34 faces (f_1-f_{34}), with its FCG shown in Figure 3(*b*). Based on edge types, features are classified as either *concave* or *convex*. The following section describes the automated detection of these features via FCG decomposition.

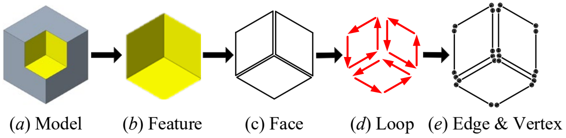
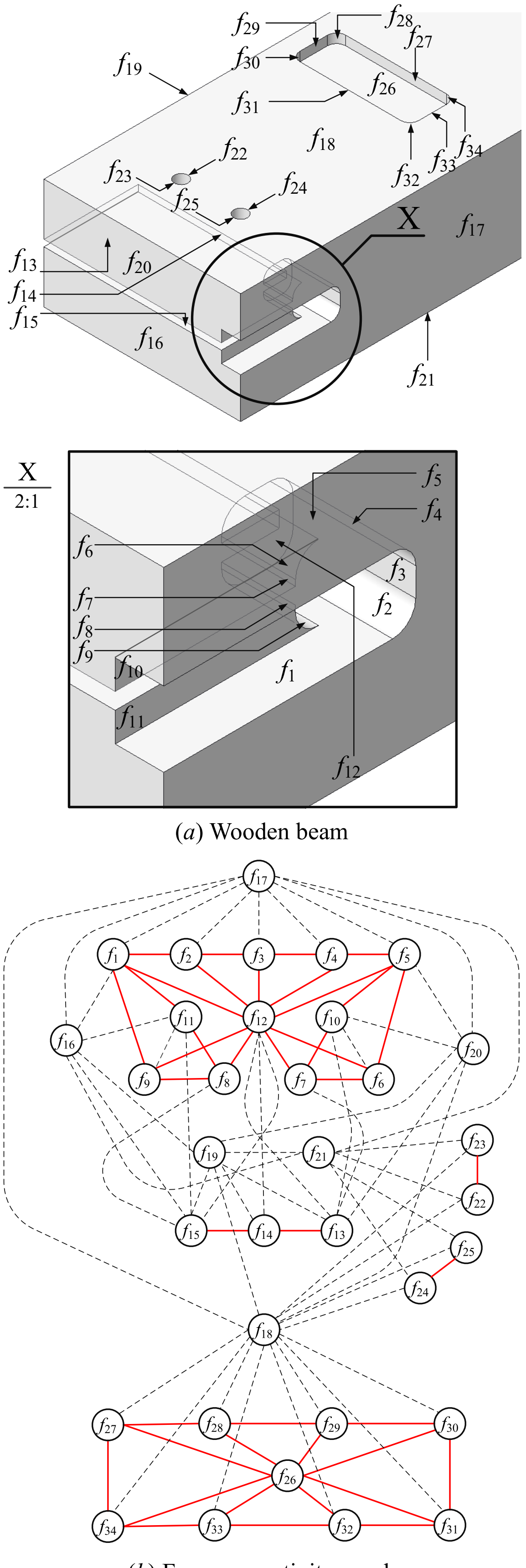


Fig. 2. Topological relationship

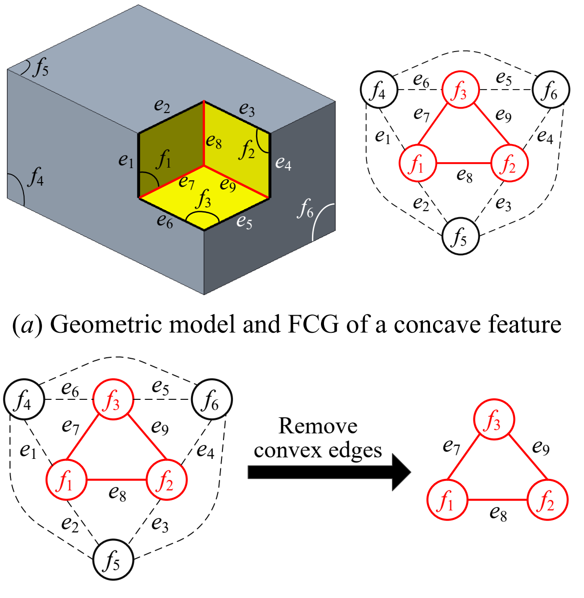


(*a*) Wooden beam

(*b*) Face connectivity graph

Fig. 3. Face connectivity graph of a wooden beam

All faces within a feature are interconnected by concave edges (solid lines), while convex edges (dashed lines) define the feature's boundary by linking to external faces. For example, in Figure 4(*a*), faces f_1 to f_3 form a concave feature: boundary edges e_1-e_6 are convex, while internal edges e_7-e_9 are concave. Based on this principle, removing all adjacent faces connected via boundary convex edges isolates the concave feature, as shown in Figure 4(*b*). The detection algorithm proceeds as follows:



(*a*) Geometric model and FCG of a concave feature

(*b*) Removal of convex edges

Fig. 4. FCG processing of a concave feature

(1) Select any concave edge from the geometric model and identify its adjacent faces. For example, in Figure 5(*a*), edge e_1 is a concave edge, and its adjacent faces are f_1 and f_2.

(2) For each adjacent face, classify the edges in its outer loop as either concave or convex. In Figure 5(*b*), face f_1 has one outer loop consisting of edges e_1, e_2, e_3, and e_4, where e_2 and e_3 are convex, and e_1 and e_4 are concave. Face f_2 has one outer loop composed of edges e_1, e_5, e_6, e_7, e_8, and e_9, where e_9 is convex, and e_1, e_5, e_6, e_7, and e_8 are concave. In summary, the convex edges are e_2, e_3, and e_9, while the concave edges are e_1, e_4, e_5, e_6, e_7, and e_8.

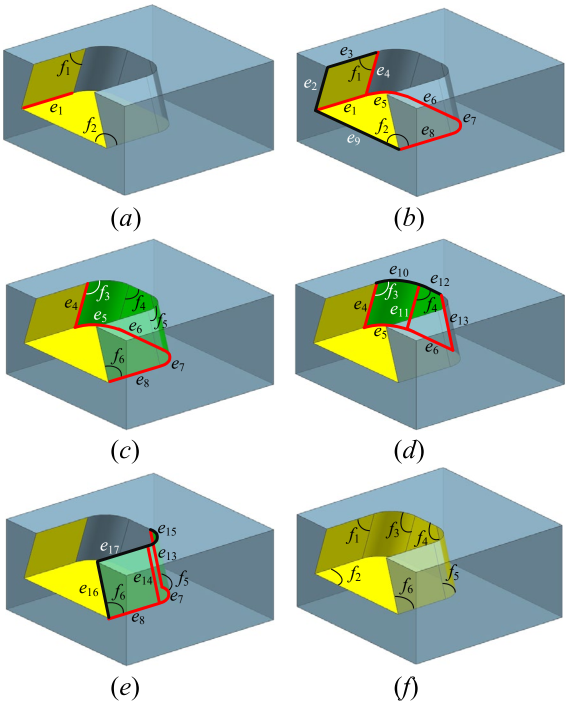
(3) Use each concave edge to find new adjacent faces. For instance, in Figure 5(*b*), edge e_1 leads to faces f_1 and f_2, which were already identified in Step (1) and are therefore not considered new. In Figure 5(*c*), using the five concave edges e_4, e_5, e_6, e_7, and e_8, four new adjacent faces, f_3, f_4, f_5, and f_6, are discovered.

(4) Repeat Steps (2) and (3) until no additional adjacent faces are found. The resulting set of faces constitutes a concave feature. As illustrated in

Figures 5(d) and 5(e), the outer loops of faces f_3, f_4, f_5, and f_6 are analyzed similarly to Step (2):

- f_3: edges e_4, e_5, e_{10}, and e_{11}; e_{10} is convex, while e_4, e_5, and e_{11} are concave.
- f_4: edges e_6, e_{11}, e_{12}, and e_{13}; e_{12} is convex, while e_6, e_{11}, and e_{13} are concave.
- f_5: edges e_7, e_{13}, e_{14}, and e_{15}; e_{15} is convex, while e_7, e_{13}, and e_{14} are concave.
- f_6: edges e_8, e_{14}, e_{16}, and e_{17}; e_{16} and e_{17} are convex, while e_8 and e_{14} are concave.

The newly identified concave edges, e_4, e_5, e_{11}, e_6, e_{13}, e_7, e_{14}, and e_8, are then used to search for additional adjacent faces as described in Step (3). Since no new faces are found, the search terminates. The final result is that faces f_1 through f_6 collectively form a concave feature, as shown in Figure 5(f).



(a) (b)

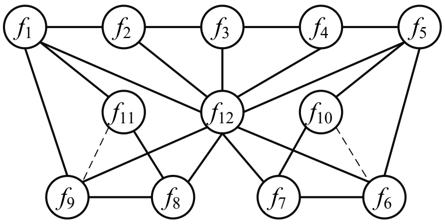
(c) (d)

(e) (f)

Fig. 5. Search process for faces in a concave feature

AUTOMATED FEATURE RECOGNITION FOR WOODEN BEAMS

The features of a wooden beam can be represented using a Face Connectivity Matrix (FCM), which encodes the adjacency relationships between faces. For example, the geometric feature shown in the detail view of Figure 3(a) comprises 12 faces, f_1 to f_{12}. Its corresponding FCG is shown in Figure 6(a), while the associated FCM is illustrated in Figure 6(b). The FCM is a symmetric matrix in which the rows and columns represent individual faces, and each element indicates the adjacency relationship between a pair of faces: a value of 1 denotes adjacency, while 0 indicates no connection.



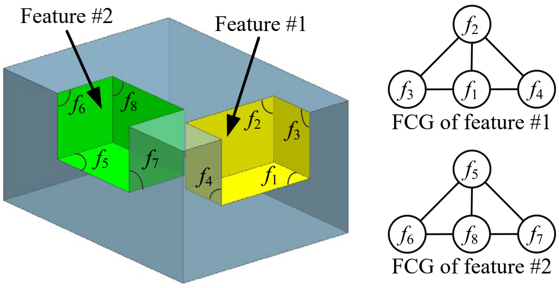
(a) Face connectivity graph

	f_1	f_2	f_3	f_4	f_5	f_6	f_7	f_8	f_9	f_{10}	f_{11}	f_{12}
f_1	0	1	0	0	0	0	0	0	1	0	1	1
f_2	1	0	1	0	0	0	0	0	0	0	0	1
f_3	0	1	0	1	0	0	0	0	0	0	0	1
f_4	0	0	1	0	1	0	0	0	0	0	0	1
f_5	0	0	0	1	0	1	0	0	0	1	0	1
f_6	0	0	0	0	1	0	1	0	0	1	0	1
f_7	0	0	0	0	0	1	0	0	0	1	0	1
f_8	0	0	0	0	0	0	0	0	1	0	1	1
f_9	1	0	0	0	0	0	0	1	0	0	1	1
f_{10}	0	0	0	0	1	1	1	0	0	0	0	0
f_{11}	1	0	0	0	0	0	0	1	1	0	0	0
f_{12}	1	1	1	1	1	1	1	1	1	0	0	0

(b) Face connectivity matrix

Fig. 6. FCG and FCM for features in Fig. 3(a)

Although identical geometric features share the same FCG, their FCMs may differ due to variations in face ordering. As shown in Figure 7(a), both feature #1 and feature #2 are blind slots and thus share the same FCG. However, their respective FCMs, Figures 7(b) and 7(c), differ because of differences in face order. Nevertheless, such FCMs can be made identical by appropriately reordering their rows and columns. For instance, as illustrated in Figure 8, swapping face f_7 with f_8, followed by swapping f_6 with f_8 in feature #2's blind slot FCM, results in an FCM identical to that of feature #1's blind slot.



(a) FCG of two blind slots

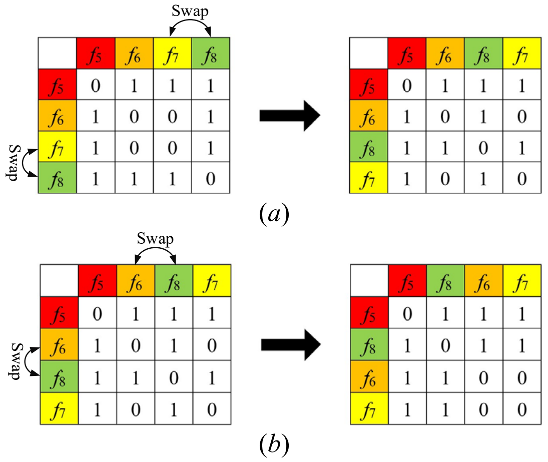
	f_1	f_2	f_3	f_4
f_1	0	1	1	1
f_2	1	0	1	1
f_3	1	1	0	0
f_4	1	1	0	0

(b) FCM of feature #1

	f_5	f_6	f_7	f_8
f_5	0	1	1	1
f_6	1	0	0	1
f_7	1	0	0	1
f_8	1	1	1	0

(c) FCM of feature #2

Fig. 7. FCG and FCM of blind slot features



$$\begin{array}{c|cccc} & f_5 & f_6 & f_7 & f_8 \\ \hline f_5 & 0 & 1 & 1 & 1 \\ f_6 & 1 & 0 & 0 & 1 \\ f_7 & 1 & 0 & 0 & 1 \\ f_8 & 1 & 1 & 1 & 0 \end{array} \rightarrow \begin{array}{c|cccc} & f_5 & f_6 & f_8 & f_7 \\ \hline f_5 & 0 & 1 & 1 & 1 \\ f_6 & 1 & 0 & 1 & 0 \\ f_8 & 1 & 1 & 0 & 1 \\ f_7 & 1 & 0 & 1 & 0 \end{array}$$

(*a*)

$$\begin{array}{c|cccc} & f_5 & f_6 & f_8 & f_7 \\ \hline f_5 & 0 & 1 & 1 & 1 \\ f_6 & 1 & 0 & 1 & 0 \\ f_8 & 1 & 1 & 0 & 1 \\ f_7 & 1 & 0 & 1 & 0 \end{array} \rightarrow \begin{array}{c|cccc} & f_5 & f_8 & f_6 & f_7 \\ \hline f_5 & 0 & 1 & 1 & 1 \\ f_8 & 1 & 0 & 1 & 1 \\ f_6 & 1 & 1 & 0 & 0 \\ f_7 & 1 & 1 & 0 & 0 \end{array}$$

(*b*)

Fig. 8. Matrix permutation of feature #2's blind slot

The reordering process consists of three main steps:

(1) Matrix segmentation: Divide the original matrix into smaller submatrices.

(2) Matrix permutation: Reorder the submatrices based on the arrangement of their elements.

(3) Feature encoding: Convert the reordered symmetric matrix into a one-dimensional array to complete the encoding process.

The subsequent sections provide a detailed explanation of each step.

Matrix segmentation

The procedure to divide an FCM into smaller submatrices is as follows:

(1) Compute and sort column sums: Calculate the sum of elements in each column of the matrix and sort them from left to right in descending order. For example, summing the elements in each column of the matrix shown in Figure 6(*b*) yields a total of 9 for face f_{12}, totals of 4 for faces f_9, f_1, f_5, and f_6, and totals of 3 for all remaining faces. Sorting these columns accordingly produces the matrix shown in Figure 9(*a*).

(2) Initial partitioning: Insert vertical and horizontal division lines between columns and rows with different sum values. Then, extract rows and columns with identical sum values to form smaller submatrices. As illustrated in Figure 9(*b*), division lines are placed between f_{12} and f_9, and between f_6 and f_2, resulting in three submatrices: a 1×1 matrix with a sum of 9, a 4×4 matrix with a sum of 4, and a 7×7 matrix with a sum of 3.

(3) Recursive sorting within submatrices: For each submatrix, calculate the sum of elements in each column and reorder the columns in descending order. For instance, in the 7×7 matrix, face f_3 has a sum of 2, while all other faces have a sum of 1. Thus, f_3 is moved to the first column, as shown in Figure 9(*c*).

(4) Recursive partitioning: Insert division lines between rows and columns with different sum values within the submatrix and extract new submatrices accordingly. In Figure 9(*d*), a division is made between f_3 and f_2, resulting in a 1×1 matrix with a sum of 2 and a 6×6 matrix with a sum of 1.

(5) Repeat until fully partitioned: Repeat steps (3) and (4) recursively until all submatrices contain rows and columns with identical sum values. For example, summing the elements in each column of the matrix in Figure 9(*e*) results in a sum of 1 for f_7, f_{11}, f_8, and f_{10}, which are placed in the first four positions. Faces f_2 and f_4 each have a sum of 0 and are placed after f_{10}. A division line is then drawn between f_{10} and f_2, producing two final submatrices: a 4×4 matrix with a sum of 1, and a 2×2 matrix with a sum of 0, as shown in Figure 9(*f*). Since the 2×2 matrix requires no further sorting or partitioning, the matrix segmentation process is complete.

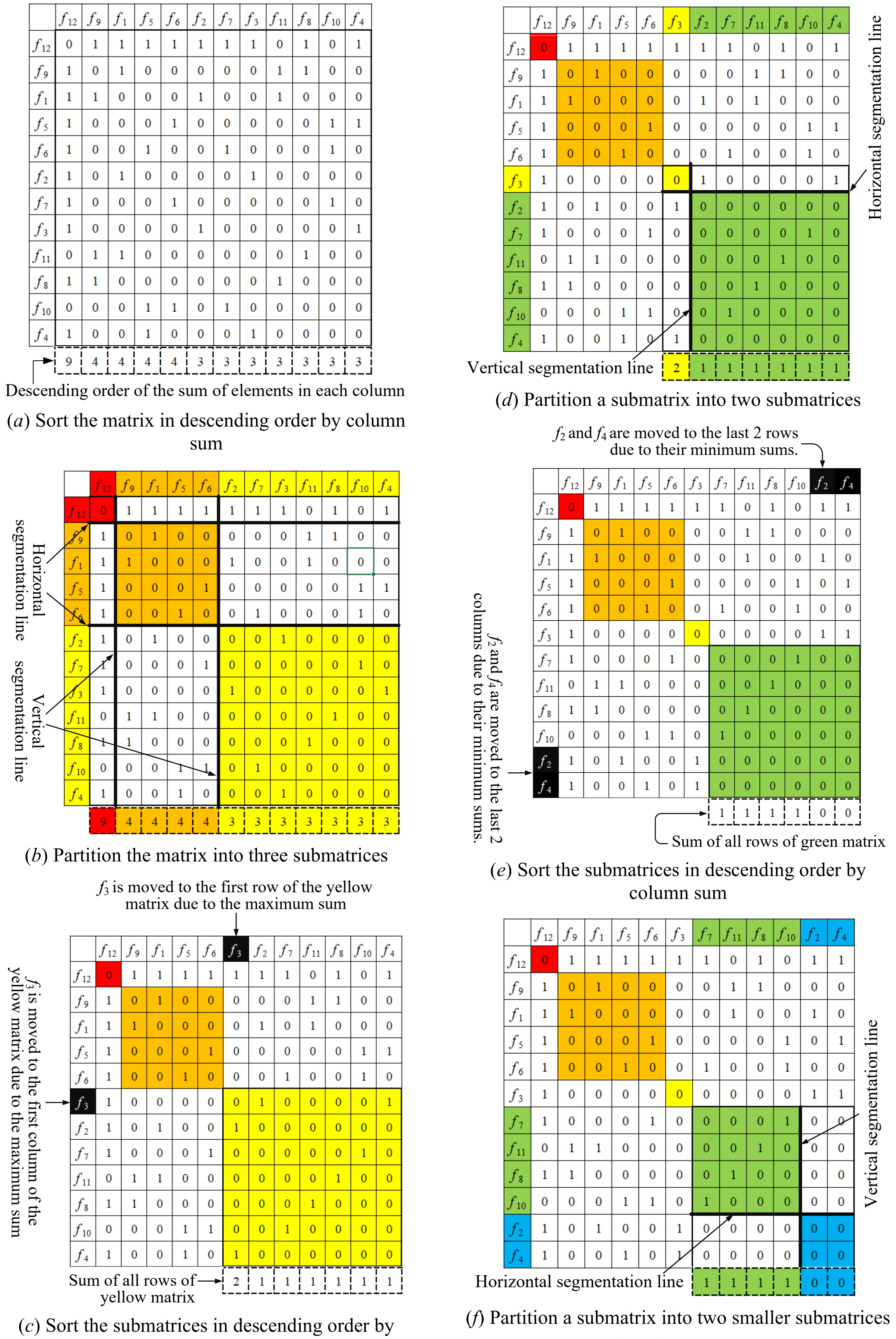
Matrix permutation

The procedure for sorting the columns and rows of each submatrix is as follows:

(1) Compare the binary values from top to bottom within the submatrix and sort the columns from left to right in descending order. For example, in the 4×4 submatrix of Figure 9(*f*), the binary value of column f_7 (accumulated from row f_7 to f_{10}) is 0001, which equals 1 in decimal; the binary value of column f_{11} is 0010, or 2 in decimal. The two columns are swapped, resulting in the matrix shown in Figure 10(*a*).

(2) Repeat step (1) until the columns and rows of the submatrix are fully sorted in descending order. Taking the 4×4 submatrix again as an example, comparing f_7 and f_8: the binary value of column f_7 (from row f_{11} to f_{10}) is 0001 (decimal 1), while column f_8 is 1000 (decimal 8). Therefore, the columns are swapped as shown in Figure 10(*b*). When binary value comparisons between two faces cause infinite swapping (as in the case shown in Figure 10(*c*)), this issue is resolved by temporarily setting all elements of both columns as zeros during comparison. For example, in the submatrix in Figure 10(*d*), columns f_{11} and f_8 are set as zero vectors to prevent infinite swapping.

(3) Compare binary values of submatrix elements, but this time the binary values are computed from the first row to the matrix elements of the two adjacent faces. Sort the columns from left to right in descending order. In the 4×4 submatrix, comparing f_{11} and f_8: the binary value of column f_{11} (from row f_{12} to f_8) is 01100000, or 96 in decimal; that of f_8 is 11000000, or 192. Thus, they are swapped as shown in Figure 10(*e*).



	f_{12}	f_9	f_1	f_5	f_6	f_2	f_7	f_3	f_{11}	f_8	f_{10}	f_4
f_{12}	0	1	1	1	1	1	1	1	0	1	0	1
f_9	1	0	1	0	0	0	0	0	1	1	0	0
f_1	1	1	0	0	0	1	0	0	1	0	0	0
f_5	1	0	0	0	1	0	0	0	0	0	1	1
f_6	1	0	0	1	0	0	1	0	0	0	1	0
f_2	1	0	1	0	0	0	0	1	0	0	0	0
f_7	1	0	0	0	1	0	0	0	0	0	1	0
f_3	1	0	0	0	0	1	0	0	0	0	0	1
f_{11}	0	1	1	0	0	0	0	0	0	1	0	0
f_8	1	1	0	0	0	0	0	0	1	0	0	0
f_{10}	0	0	0	1	1	0	1	0	0	0	0	0
f_4	1	0	0	1	0	0	0	1	0	0	0	0
	9	4	4	4	4	3	3	3	3	3	3	3

Descending order of the sum of elements in each column

(*a*) Sort the matrix in descending order by column sum

	f_{12}	f_9	f_1	f_5	f_6	f_2	f_7	f_3	f_{11}	f_8	f_{10}	f_4
f_{12}	0	1	1	1	1	1	1	1	0	1	0	1
f_9	1	0	1	0	0	0	0	0	1	1	0	0
f_1	1	1	0	0	0	1	0	0	1	0	0	0
f_5	1	0	0	0	1	0	0	0	0	0	1	1
f_6	1	0	0	1	0	0	1	0	0	0	1	0
f_2	1	0	1	0	0	0	0	1	0	0	0	0
f_7	1	0	0	0	1	0	0	0	0	0	1	0
f_3	1	0	0	0	0	1	0	0	0	0	0	1
f_{11}	0	1	1	0	0	0	0	0	0	1	0	0
f_8	1	1	0	0	0	0	0	0	1	0	0	0
f_{10}	0	0	0	1	1	0	1	0	0	0	0	0
f_4	1	0	0	1	0	0	0	1	0	0	0	0
	9	4	4	4	4	3	3	3	3	3	3	3

Horizontal segmentation line

Vertical segmentation line

(*b*) Partition the matrix into three submatrices

f_3 is moved to the first row of the yellow matrix due to the maximum sum

f_3 is moved to the first column of the yellow matrix due to the maximum sum

	f_{12}	f_9	f_1	f_5	f_6	f_3	f_2	f_7	f_{11}	f_8	f_{10}	f_4
f_{12}	0	1	1	1	1	1	1	1	0	1	0	1
f_9	1	0	1	0	0	0	0	0	1	1	0	0
f_1	1	1	0	0	0	0	1	0	1	0	0	0
f_5	1	0	0	0	1	0	0	0	0	0	1	1
f_6	1	0	0	1	0	0	0	1	0	0	1	0
f_3	1	0	0	0	0	0	1	0	0	0	0	1
f_2	1	0	1	0	0	1	0	0	0	0	0	0
f_7	1	0	0	0	1	0	0	0	0	0	1	0
f_{11}	0	1	1	0	0	0	0	0	0	1	0	0
f_8	1	1	0	0	0	0	0	0	1	0	0	0
f_{10}	0	0	0	1	1	0	0	1	0	0	0	0
f_4	1	0	0	1	0	1	0	0	0	0	0	0
						2	1	1	1	1	1	1

Sum of all rows of yellow matrix

(*c*) Sort the submatrices in descending order by column sum

	f_{12}	f_9	f_1	f_5	f_6	f_3	f_2	f_7	f_{11}	f_8	f_{10}	f_4
f_{12}	0	1	1	1	1	1	1	1	0	1	0	1
f_9	1	0	1	0	0	0	0	0	1	1	0	0
f_1	1	1	0	0	0	0	1	0	1	0	0	0
f_5	1	0	0	0	1	0	0	0	0	0	1	1
f_6	1	0	0	1	0	0	0	1	0	0	1	0
f_3	1	0	0	0	0	0	1	0	0	0	0	1
f_2	1	0	1	0	0	1	0	0	0	0	0	0
f_7	1	0	0	0	1	0	0	0	0	0	1	0
f_{11}	0	1	1	0	0	0	0	0	0	1	0	0
f_8	1	1	0	0	0	0	0	0	1	0	0	0
f_{10}	0	0	0	1	1	0	0	1	0	0	0	0
f_4	1	0	0	1	0	1	0	0	0	0	0	0
						2	1	1	1	1	1	1

Horizontal segmentation line

Vertical segmentation line

(*d*) Partition a submatrix into two submatrices

f_2 and f_4 are moved to the last 2 rows due to their minimum sums.

f_2 and f_4 are moved to the last 2 columns due to their minimum sums.

	f_{12}	f_9	f_1	f_5	f_6	f_3	f_7	f_{11}	f_8	f_{10}	f_2	f_4
f_{12}	0	1	1	1	1	1	1	0	1	0	1	1
f_9	1	0	1	0	0	0	0	1	1	0	0	0
f_1	1	1	0	0	0	0	0	1	0	0	1	0
f_5	1	0	0	0	1	0	0	0	0	1	0	1
f_6	1	0	0	1	0	0	1	0	0	1	0	0
f_3	1	0	0	0	0	0	0	0	0	0	1	1
f_7	1	0	0	0	1	0	0	0	0	1	0	0
f_{11}	0	1	1	0	0	0	0	0	1	0	0	0
f_8	1	1	0	0	0	0	0	1	0	0	0	0
f_{10}	0	0	0	1	1	0	1	0	0	0	0	0
f_2	1	0	1	0	0	1	0	0	0	0	0	0
f_4	1	0	0	1	0	1	0	0	0	0	0	0
							1	1	1	1	0	0

Sum of all rows of green matrix

(*e*) Sort the submatrices in descending order by column sum

	f_{12}	f_9	f_1	f_5	f_6	f_3	f_7	f_{11}	f_8	f_{10}	f_2	f_4
f_{12}	0	1	1	1	1	1	1	0	1	0	1	1
f_9	1	0	1	0	0	0	0	1	1	0	0	0
f_1	1	1	0	0	0	0	0	1	0	0	1	0
f_5	1	0	0	0	1	0	0	0	0	1	0	1
f_6	1	0	0	1	0	0	1	0	0	1	0	0
f_3	1	0	0	0	0	0	0	0	0	0	1	1
f_7	1	0	0	0	1	0	0	0	0	1	0	0
f_{11}	0	1	1	0	0	0	0	0	1	0	0	0
f_8	1	1	0	0	0	0	0	1	0	0	0	0
f_{10}	0	0	0	1	1	0	1	0	0	0	0	0
f_2	1	0	1	0	0	1	0	0	0	0	0	0
f_4	1	0	0	1	0	1	0	0	0	0	0	0
							1	1	1	1	0	0

Vertical segmentation line

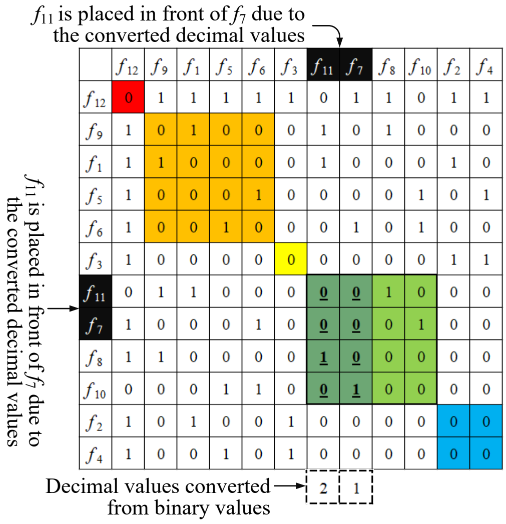
Horizontal segmentation line

(*f*) Partition a submatrix into two smaller submatrices

Fig. 9. Procedure for matrix segmentation

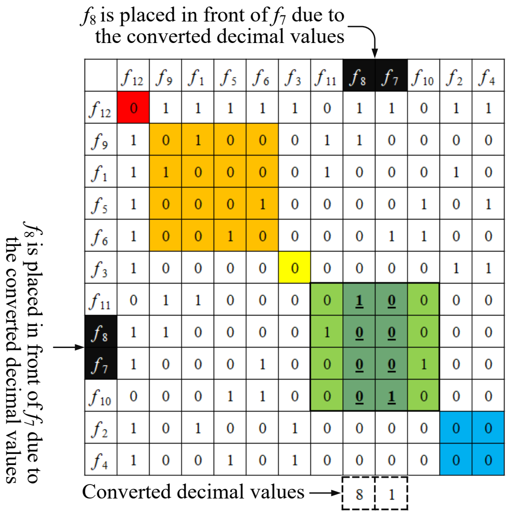
(4) Repeat step (3) until the columns and rows of the submatrix are fully sorted in descending order. For example, when comparing f_{11} and f_7 in the 4×4 submatrix: column f_{11} (from row f_{12} to f_7) has a binary value of 011000100 (decimal 196), while f_7 has 100010000 (decimal 272). The two columns are then swapped, as in Figure 10(*f*). At this point, all elements in the 4×4 submatrix are fully rearranged, so step (3) is no longer needed.

(5) Compare the binary values of submatrix elements again, but this time the binary values are calculated from the matrix elements of the two adjacent faces to the last row. Sort the columns from left to right in descending order. For the 4×4 submatrix in Figure 10(*f*), comparing f_5 and f_6: column f_5 (from row f_5 to f_4) has a binary value of 000000101 (decimal 5), and f_6 has 000010100 (decimal 20). Therefore, the columns are swapped, as shown in Figure 10(*g*).


f_{11} is placed in front of f_7 due to the converted decimal values
f_{11} is placed in front of f_7 due to the converted decimal values
Decimal values converted from binary values

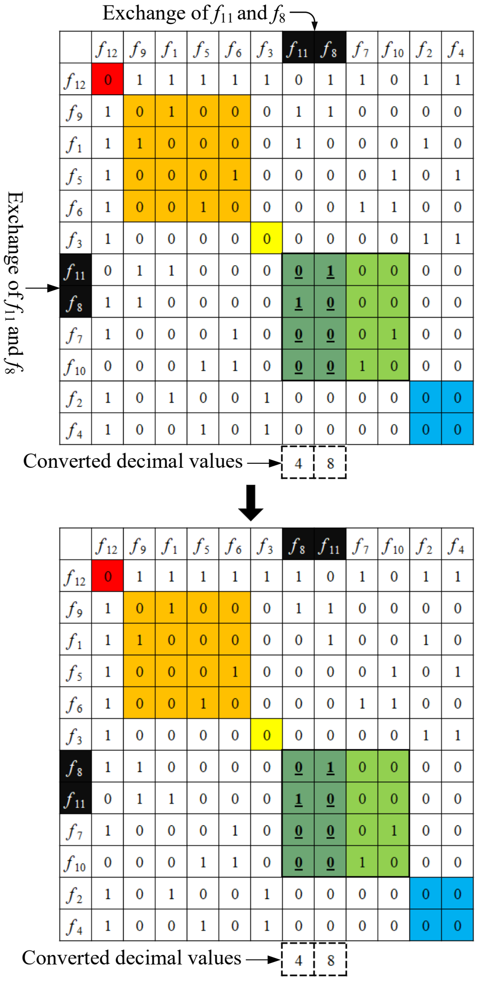
	f_{12}	f_9	f_1	f_5	f_6	f_3	f_{11}	f_7	f_8	f_{10}	f_2	f_4
f_{12}	0	1	1	1	1	1	0	1	1	0	1	1
f_9	1	0	1	0	0	0	1	0	1	0	0	0
f_1	1	1	0	0	0	0	1	0	0	0	1	0
f_5	1	0	0	0	1	0	0	0	0	1	0	1
f_6	1	0	0	1	0	0	0	1	0	1	0	0
f_3	1	0	0	0	0	0	0	0	0	0	1	1
f_{11}	0	1	1	0	0	0	0	0	1	0	0	0
f_7	1	0	0	0	1	0	0	0	0	1	0	0
f_8	1	1	0	0	0	0	1	0	0	0	0	0
f_{10}	0	0	0	1	1	0	0	1	0	0	0	0
f_2	1	0	1	0	0	1	0	0	0	0	0	0
f_4	1	0	0	1	0	1	0	0	0	0	0	0
							2	1				

(*a*) Swap column f_7 with column f_{10}


f_8 is placed in front of f_7 due to the converted decimal values
f_8 is placed in front of f_7 due to the converted decimal values
Converted decimal values

	f_{12}	f_9	f_1	f_5	f_6	f_3	f_{11}	f_8	f_7	f_{10}	f_2	f_4
f_{12}	0	1	1	1	1	1	0	1	1	0	1	1
f_9	1	0	1	0	0	0	1	1	0	0	0	0
f_1	1	1	0	0	0	0	1	0	0	0	1	0
f_5	1	0	0	0	1	0	0	0	0	1	0	1
f_6	1	0	0	1	0	0	0	0	1	1	0	0
f_3	1	0	0	0	0	0	0	0	0	0	1	1
f_{11}	0	1	1	0	0	0	0	1	0	0	0	0
f_8	1	1	0	0	0	0	1	0	0	0	0	0
f_7	1	0	0	0	1	0	0	0	0	1	0	0
f_{10}	0	0	0	1	1	0	0	0	1	0	0	0
f_2	1	0	1	0	0	1	0	0	0	0	0	0
f_4	1	0	0	1	0	1	0	0	0	0	0	0
								8	1			

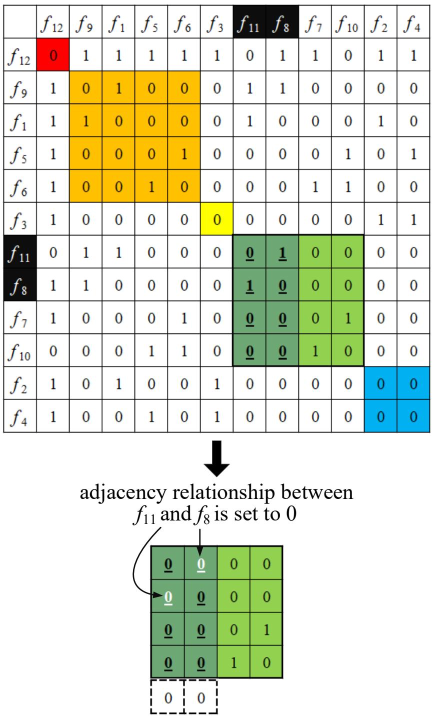
(*b*) Swap column f_7 with column f_8


Exchange of f_{11} and f_8
Exchange of f_{11} and f_8
Converted decimal values
Converted decimal values

	f_{12}	f_9	f_1	f_5	f_6	f_3	f_{11}	f_8	f_7	f_{10}	f_2	f_4
f_{12}	0	1	1	1	1	1	0	1	1	0	1	1
f_9	1	0	1	0	0	0	1	1	0	0	0	0
f_1	1	1	0	0	0	0	1	0	0	0	1	0
f_5	1	0	0	0	1	0	0	0	0	1	0	1
f_6	1	0	0	1	0	0	0	0	1	1	0	0
f_3	1	0	0	0	0	0	0	0	0	0	1	1
f_{11}	0	1	1	0	0	0	0	1	0	0	0	0
f_8	1	1	0	0	0	0	1	0	0	0	0	0
f_7	1	0	0	0	1	0	0	0	0	1	0	0
f_{10}	0	0	0	1	1	0	0	0	1	0	0	0
f_2	1	0	1	0	0	1	0	0	0	0	0	0
f_4	1	0	0	1	0	1	0	0	0	0	0	0
							4	8				

	f_{12}	f_9	f_1	f_5	f_6	f_3	f_8	f_{11}	f_7	f_{10}	f_2	f_4
f_{12}	0	1	1	1	1	1	1	0	1	0	1	1
f_9	1	0	1	0	0	0	1	1	0	0	0	0
f_1	1	1	0	0	0	0	0	1	0	0	1	0
f_5	1	0	0	0	1	0	0	0	0	1	0	1
f_6	1	0	0	1	0	0	0	0	1	1	0	0
f_3	1	0	0	0	0	0	0	0	0	0	1	1
f_8	1	1	0	0	0	0	0	1	0	0	0	0
f_{11}	0	1	1	0	0	0	1	0	0	0	0	0
f_7	1	0	0	0	1	0	0	0	0	1	0	0
f_{10}	0	0	0	1	1	0	0	0	1	0	0	0
f_2	1	0	1	0	0	1	0	0	0	0	0	0
f_4	1	0	0	1	0	1	0	0	0	0	0	0
							4	8				

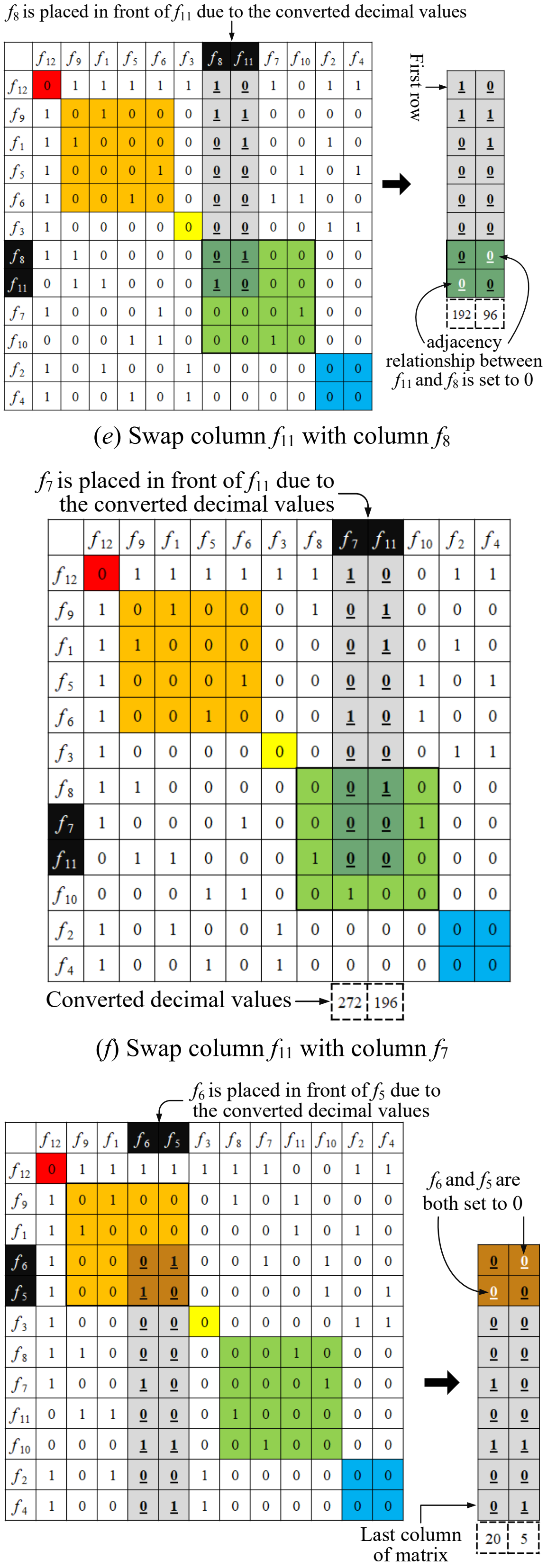
(*c*) Example of iterative swapping operations


adjacency relationship between f_{11} and f_8 is set to 0

	f_{12}	f_9	f_1	f_5	f_6	f_3	f_{11}	f_8	f_7	f_{10}	f_2	f_4
f_{12}	0	1	1	1	1	1	0	1	1	0	1	1
f_9	1	0	1	0	0	0	1	1	0	0	0	0
f_1	1	1	0	0	0	0	1	0	0	0	1	0
f_5	1	0	0	0	1	0	0	0	0	1	0	1
f_6	1	0	0	1	0	0	0	0	1	1	0	0
f_3	1	0	0	0	0	0	0	0	0	0	1	1
f_{11}	0	1	1	0	0	0	0	1	0	0	0	0
f_8	1	1	0	0	0	0	1	0	0	0	0	0
f_7	1	0	0	0	1	0	0	0	0	1	0	0
f_{10}	0	0	0	1	1	0	0	0	1	0	0	0
f_2	1	0	1	0	0	1	0	0	0	0	0	0
f_4	1	0	0	1	0	1	0	0	0	0	0	0

0	0	0	0
0	0	0	0
0	0	0	1
0	0	1	0
0	0		

(*d*) Set the matrix elements of f_{11} and f_8 as zero for comparison



f_8 is placed in front of f_{11} due to the converted decimal values

	f_{12}	f_9	f_1	f_5	f_6	f_3	f_8	f_{11}	f_7	f_{10}	f_2	f_4
f_{12}	0	1	1	1	1	1	1	0	1	0	1	1
f_9	1	0	1	0	0	0	1	1	0	0	0	0
f_1	1	1	0	0	0	0	0	1	0	0	1	0
f_5	1	0	0	0	1	0	0	0	0	1	0	1
f_6	1	0	0	1	0	0	0	0	1	1	0	0
f_3	1	0	0	0	0	0	0	0	0	0	1	1
f_8	1	1	0	0	0	0	0	1	0	0	0	0
f_{11}	0	1	1	0	0	0	1	0	0	0	0	0
f_7	1	0	0	0	1	0	0	0	0	1	0	0
f_{10}	0	0	0	1	1	0	0	0	1	0	0	0
f_2	1	0	1	0	0	1	0	0	0	0	0	0
f_4	1	0	0	1	0	1	0	0	0	0	0	0

First row

1	0
1	1
0	1
0	0
0	0
0	0
0	0
0	0
192	96

adjacency relationship between f_{11} and f_8 is set to 0

(*e*) Swap column f_{11} with column f_8

f_7 is placed in front of f_{11} due to the converted decimal values

	f_{12}	f_9	f_1	f_5	f_6	f_3	f_8	f_7	f_{11}	f_{10}	f_2	f_4
f_{12}	0	1	1	1	1	1	1	1	0	0	1	1
f_9	1	0	1	0	0	0	1	0	1	0	0	0
f_1	1	1	0	0	0	0	0	0	1	0	1	0
f_5	1	0	0	0	1	0	0	0	0	1	0	1
f_6	1	0	0	1	0	0	0	1	0	1	0	0
f_3	1	0	0	0	0	0	0	0	0	0	1	1
f_8	1	1	0	0	0	0	0	0	1	0	0	0
f_7	1	0	0	0	1	0	0	0	0	1	0	0
f_{11}	0	1	1	0	0	0	1	0	0	0	0	0
f_{10}	0	0	0	1	1	0	0	1	0	0	0	0
f_2	1	0	1	0	0	1	0	0	0	0	0	0
f_4	1	0	0	1	0	1	0	0	0	0	0	0
Converted decimal values								272	196			

(*f*) Swap column f_{11} with column f_7

f_6 is placed in front of f_5 due to the converted decimal values

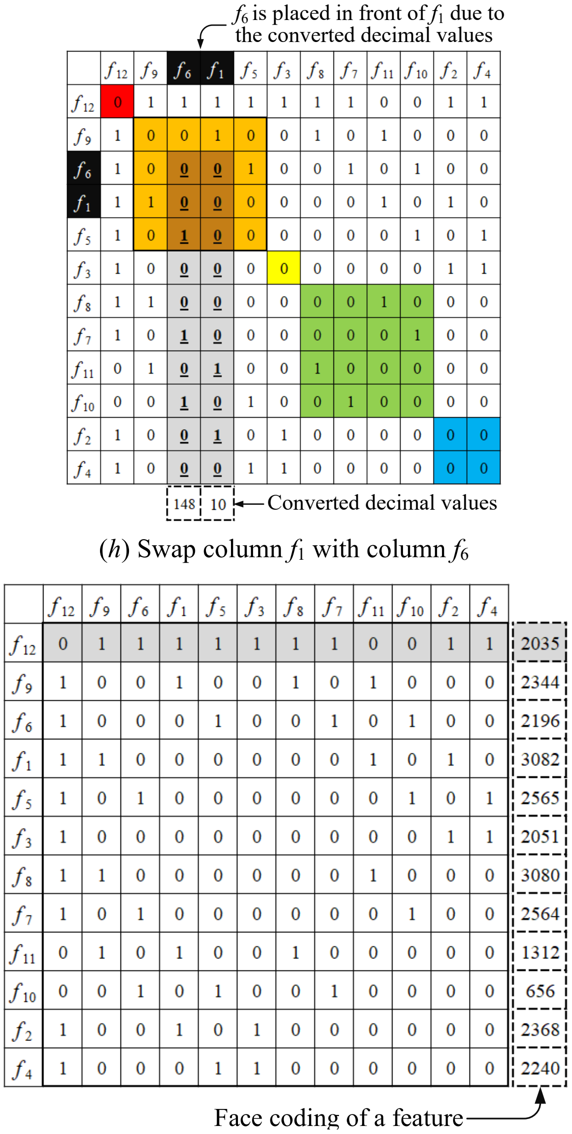
	f_{12}	f_9	f_1	f_6	f_5	f_3	f_8	f_7	f_{11}	f_{10}	f_2	f_4
f_{12}	0	1	1	1	1	1	1	1	0	0	1	1
f_9	1	0	1	0	0	0	1	0	1	0	0	0
f_1	1	1	0	0	0	0	0	0	1	0	1	0
f_6	1	0	0	0	1	0	0	1	0	1	0	0
f_5	1	0	0	1	0	0	0	0	0	1	0	1
f_3	1	0	0	0	0	0	0	0	0	0	1	1
f_8	1	1	0	0	0	0	0	0	1	0	0	0
f_7	1	0	0	1	0	0	0	0	0	1	0	0
f_{11}	0	1	1	0	0	0	1	0	0	0	0	0
f_{10}	0	0	0	1	1	0	0	1	0	0	0	0
f_2	1	0	1	0	0	1	0	0	0	0	0	0
f_4	1	0	0	0	1	1	0	0	0	0	0	0

f_6 and f_5 are both set to 0

0	0
0	0
0	0
0	0
1	0
0	0
1	1
0	0
0	1
20	5

Last column of matrix

(g) Swap column f_5 with column f_6



f_6 is placed in front of f_1 due to the converted decimal values

	f_{12}	f_9	f_6	f_1	f_5	f_3	f_8	f_7	f_{11}	f_{10}	f_2	f_4
f_{12}	0	1	1	1	1	1	1	1	0	0	1	1
f_9	1	0	0	1	0	0	1	0	1	0	0	0
f_6	1	0	0	0	1	0	0	1	0	1	0	0
f_1	1	1	0	0	0	0	0	0	1	0	1	0
f_5	1	0	1	0	0	0	0	0	0	1	0	1
f_3	1	0	0	0	0	0	0	0	0	0	1	1
f_8	1	1	0	0	0	0	0	0	1	0	0	0
f_7	1	0	1	0	0	0	0	0	0	1	0	0
f_{11}	0	1	0	1	0	0	1	0	0	0	0	0
f_{10}	0	0	1	0	1	0	0	1	0	0	0	0
f_2	1	0	0	1	0	1	0	0	0	0	0	0
f_4	1	0	0	0	1	1	0	0	0	0	0	0
Converted decimal values			148	10								

(*h*) Swap column f_1 with column f_6

	f_{12}	f_9	f_6	f_1	f_5	f_3	f_8	f_7	f_{11}	f_{10}	f_2	f_4	Face coding of a feature
f_{12}	0	1	1	1	1	1	1	1	0	0	1	1	2035
f_9	1	0	0	1	0	0	1	0	1	0	0	0	2344
f_6	1	0	0	0	1	0	0	1	0	1	0	0	2196
f_1	1	1	0	0	0	0	0	0	1	0	1	0	3082
f_5	1	0	1	0	0	0	0	0	0	1	0	1	2565
f_3	1	0	0	0	0	0	0	0	0	0	1	1	2051
f_8	1	1	0	0	0	0	0	0	1	0	0	0	3080
f_7	1	0	1	0	0	0	0	0	0	1	0	0	2564
f_{11}	0	1	0	1	0	0	1	0	0	0	0	0	1312
f_{10}	0	0	1	0	1	0	0	1	0	0	0	0	656
f_2	1	0	0	1	0	1	0	0	0	0	0	0	2368
f_4	1	0	0	0	1	1	0	0	0	0	0	0	2240

Face coding of a feature

(*i*) Encoding after converting the symmetric matrix into a one-dimensional array

Fig. 10. Procedure for matrix permutation

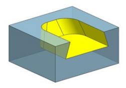
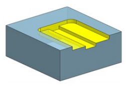
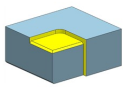
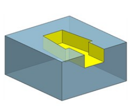
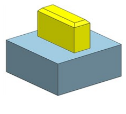
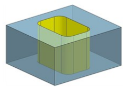
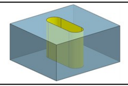
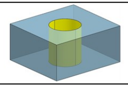
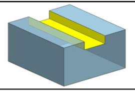
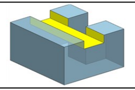
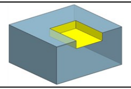
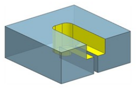
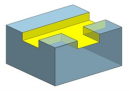
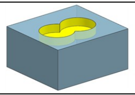
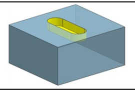
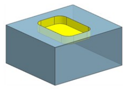
(6) Repeat step (5) until the columns and rows of the submatrix are fully sorted in descending order. For instance, in the 4×4 submatrix, comparing f_1 and f_6: column f_1 (from row f_1 to f_4) has a binary value of 000001010 (decimal 10), while f_6 has 0010010100 (decimal 148). The columns are then swapped, as shown in Figure 10(*h*). At this point, all elements in the submatrix are fully rearranged, so step (5) is no longer needed.

Feature encoding

Through the aforementioned steps, the elements of the FCM are reordered. To distinguish different features, the final step involves converting the resulting FCM into a one-dimensional array whose size corresponds to the total number of faces in the feature. Each element of this array represents the decimal value obtained by converting the cumulative binary values from the first to the last row for each face. As shown in Figure 10(*i*), taking the first element of the one-dimensional array as an example, its binary value is 011111110011, which corresponds to the decimal value 2035.

The feature database listed in Table 1 stores all beam features using feature encoding for recognition purposes.

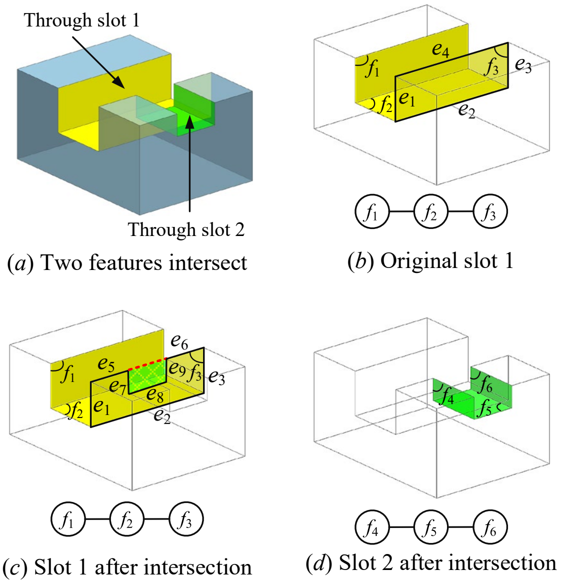
Table 1. Feature codings

Feature name	Feature geometry	Feature coding
Blind step, type I		$\begin{bmatrix}31\\38\\37\\56\\48\\40\end{bmatrix}$
Blind step, type II		$\begin{bmatrix}508\\554\\533\\560\\832\\704\\770\\641\\264\\132\end{bmatrix}$
Blind step, type III		$\begin{bmatrix}62\\85\\99\\70\\104\\88\\48\end{bmatrix}$
Blind step, type IV		$\begin{bmatrix}127\\138\\133\\140\\208\\176\\192\\160\end{bmatrix}$
Rectangular boss		$\begin{bmatrix}216\\308\\306\\209\\480\\262\\137\\73\\38\end{bmatrix}$
Rounded rectangular hole		$\begin{bmatrix}96\\144\\136\\68\\34\\17\\9\\6\end{bmatrix}$
Racetrack hole		$\begin{bmatrix}6\\9\\9\\6\end{bmatrix}$
Circular hole		$\begin{bmatrix}1\\2\end{bmatrix}$
Through slot, type I		$\begin{bmatrix}3\\4\\4\end{bmatrix}$
Through slot, type II		$\begin{bmatrix}7\\8\\8\\8\end{bmatrix}$
Blind slot		$\begin{bmatrix}7\\11\\12\\12\end{bmatrix}$
Blind step, type V		$\begin{bmatrix}2035\\2344\\2196\\3082\\2565\\2051\\3080\\2564\\1312\\656\\2368\\2240\end{bmatrix}$
Blind step, type VI		$\begin{bmatrix}31\\40\\48\\34\\36\\32\end{bmatrix}$
Circular pocket		$\begin{bmatrix}3\\5\\6\end{bmatrix}$
Racetrack pocket		$\begin{bmatrix}15\\22\\25\\25\\22\end{bmatrix}$
Rounded rectangular pocket		$\begin{bmatrix}255\\352\\400\\392\\324\\290\\273\\265\\232\end{bmatrix}$

Handling intersecting features

When the geometries of multiple features intersect, the topological and geometric relationships among those features are altered, resulting in changes to their feature encodings. For example, when two through slots intersect, one of the following three scenarios may occur:

(1) A face of a feature is trimmed: In this case, the shape and geometric dimensions of the affected face and its edges are modified, but the adjacency relationships between that face and its neighboring faces remain unchanged. As a result, the FCG of the intersected feature remains the same as that of the original feature. For instance, in Figure 11(*a*), through slot 1 is composed of faces f_1, f_2, and f_3 (see Figure 11(*b*)). When through slot 2 intersects through slot 1, as shown in Figure 11(*c*), face f_3 of through slot 1 is trimmed by the contour of through slot 2, causing edge e_4 to split into e_5 and e_6. However, the adjacency edge e_2 between faces f_3 and f_2 remains intact, preserving the connectivity among faces f_1, f_2, and f_3. Consequently, the FCG of through slot 1 remains unchanged before and after the intersection. Furthermore, since through slot 2 is unaffected by the intersection, its FCG also remains identical to that of the standard feature, as illustrated in Figure 11(*d*). This indicates that the part contains two independent features, through slot 1 and through slot 2, both classified as type I.



(*a*) Two features intersect

(*b*) Original slot 1

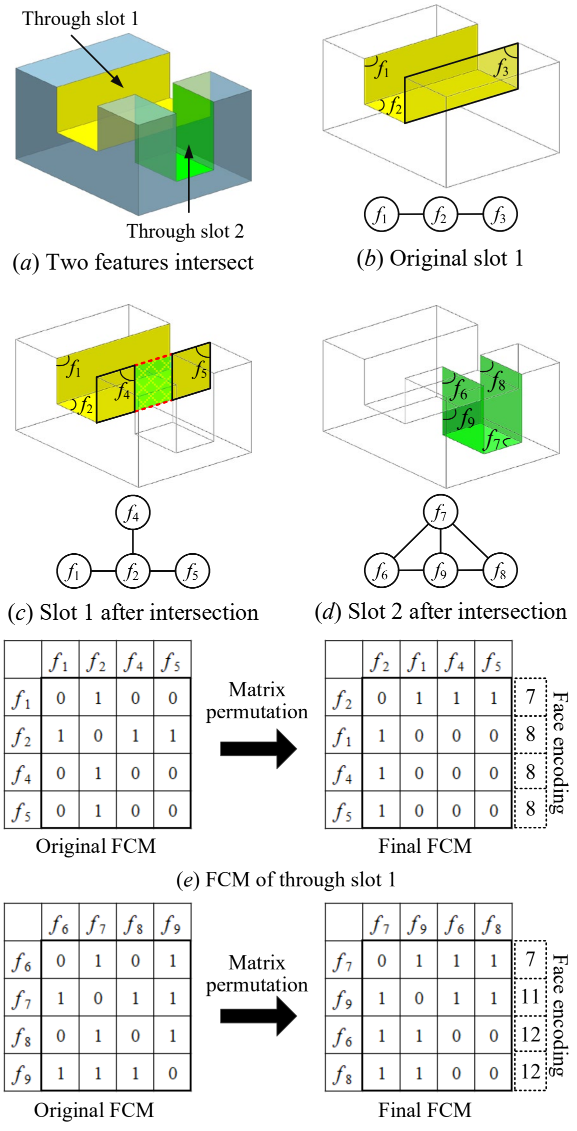
(*c*) Slot 1 after intersection

(*d*) Slot 2 after intersection

Fig. 11. First type of intersecting feature

(2) A face of a feature is divided: When a face is split into two or more faces, its topological relationships are disrupted, resulting in an FCG that differs from that of the standard feature. For example, in Figure 12(*a*), Through Slot 1 consists of faces f_1, f_2, and f_3 (see Figure 12(*b*)). When Through Slot 2 intersects Through Slot 1, face f_3 is divided into faces f_4 and f_5 (see Figure 12(*c*)), indicating that the FCG of Through Slot 1 changes after the intersection. Moreover, as Through Slot 2 does not fully penetrate the original model after dividing Through Slot 1, it intersects with the original model at face f_9, thereby altering the topological relationships of Through Slot 2, as shown in Figure 12(*d*). By generating FCMs from the FCGs of these two features and applying matrix permutation (Figures 14(*e*) and (*f*)), feature encoding is performed. The resulting encodings are [7 8 8 8] for Through Slot 1 and [7 11 12 12] for Through Slot 2. Comparing these encodings with the feature database confirms the presence of two features: Through Slot 1 is classified as a

Through Slot, Type II, and Through Slot 2 as a Blind Slot.



(*a*) Two features intersect (*b*) Original slot 1

(*c*) Slot 1 after intersection (*d*) Slot 2 after intersection

Original FCM:

	f_1	f_2	f_4	f_5
f_1	0	1	0	0
f_2	1	0	1	1
f_4	0	1	0	0
f_5	0	1	0	0

Final FCM:

	f_2	f_1	f_4	f_5	Face encoding
f_2	0	1	1	1	7
f_1	1	0	0	0	8
f_4	1	0	0	0	8
f_5	1	0	0	0	8

(*e*) FCM of through slot 1

Original FCM:

	f_6	f_7	f_8	f_9
f_6	0	1	0	1
f_7	1	0	1	1
f_8	0	1	0	1
f_9	1	1	1	0

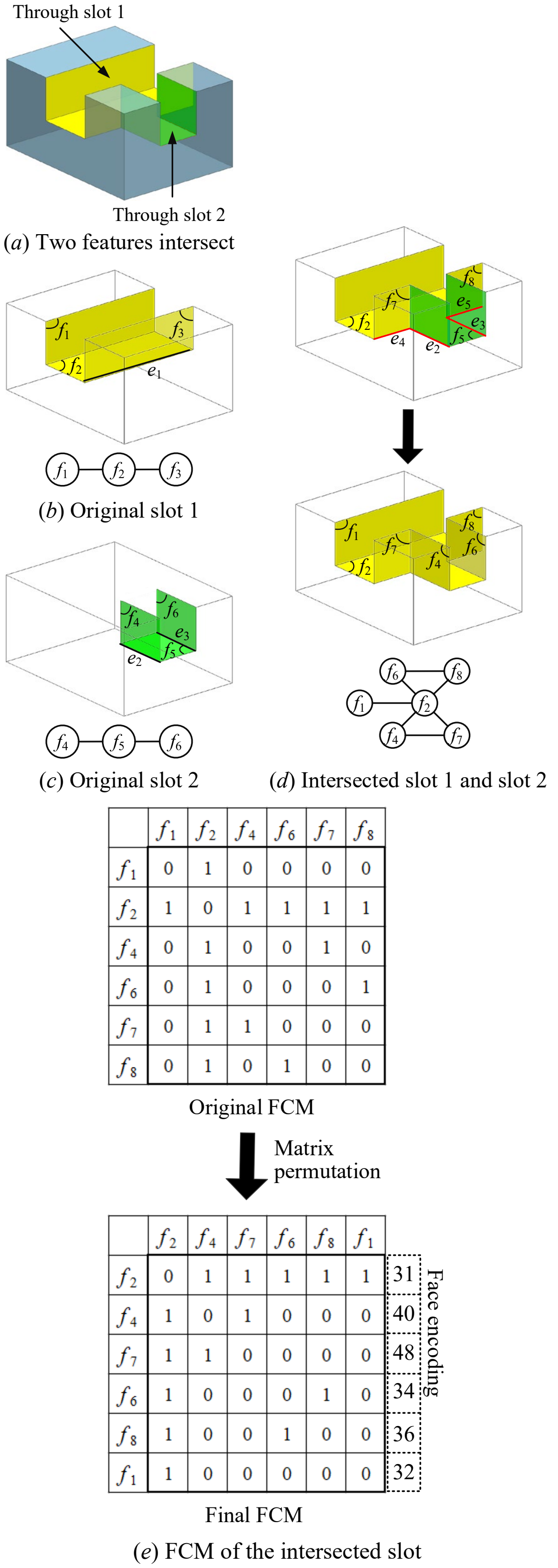
Final FCM:

	f_7	f_9	f_6	f_8	Face encoding
f_7	0	1	1	1	7
f_9	1	0	1	1	11
f_6	1	1	0	0	12
f_8	1	1	0	0	12

(*f*) FCM of through slot 2

Fig. 12. Second type of intersecting feature

(3) Adjacent concave edges of two features are connected: When two features intersect and their adjacent concave edges are connected, the associated faces of both features are identified collectively, leading to their recognition as a single feature. For example, in Figure 13(*a*), Through Slot 1 comprises faces f_1, f_2, and f_3, with edge e_1 serving as the adjacency edge between faces f_2 and f_3 (see Figure 13(*b*)). Through Slot 2 comprises faces f_4, f_5, and f_6, with edge e_2 connecting faces f_4 and f_5, and edge e_3 connecting faces f_5 and f_6 (see Figure 13(*c*)). Upon the intersection of Through Slot 2 with Through Slot 1, face f_3 of Through Slot 1 is divided into faces f_7 and f_8, resulting in edge e_1 being split into edges e_4 and e_5 (see Figure 13(*d*)). Concurrently, edge e_4 connects with edge e_2, and edge e_5 with edge e_3, indicating that Through Slot 1 and Through Slot 2 are now topologically linked and thus regarded as a single feature. Moreover, since faces f_2 and f_5 lie



(*a*) Two features intersect

(*b*) Original slot 1

(*c*) Original slot 2 (*d*) Intersected slot 1 and slot 2

Original FCM:

	f_1	f_2	f_4	f_6	f_7	f_8
f_1	0	1	0	0	0	0
f_2	1	0	1	1	1	1
f_4	0	1	0	0	1	0
f_6	0	1	0	0	0	1
f_7	0	1	1	0	0	0
f_8	0	1	0	1	0	0

Final FCM:

	f_2	f_4	f_7	f_6	f_8	f_1	Face encoding
f_2	0	1	1	1	1	1	31
f_4	1	0	1	0	0	0	40
f_7	1	1	0	0	0	0	48
f_6	1	0	0	0	1	0	34
f_8	1	0	0	1	0	0	36
f_1	1	0	0	0	0	0	32

(*e*) FCM of the intersected slot

Fig. 13. Third type of intersecting feature

on the same height, they are merged into face f_2. By generating the FCM from the feature's FCG and applying matrix permutation, as shown in Figure 13(*e*), feature encoding is performed. The resulting encoding for the combined feature is [31 40 48 34 36 32]. A comparison with the feature database indicates that the part contains a single feature, classified as a Blind Step, Type VI.

From the above three cases, it can be concluded that the proposed matrix permutation and feature encoding methods in this study are effective in accurately recognizing intersecting features.

MACHINING SEQUENCE PLANNING FOR WOODEN BEAM MACHINING

During beam machining, to avoid collisions between the machine's tool arm and the workpiece, it is necessary to adopt different tool orientations for accessing bottom features located at various positions. In this study, a reference plane perpendicular to the *X*-axis is established at the center of the beam to divide the model into left and right halves. Bottom features are classified according to their location on either side. If a feature spans both sides, it is segmented into two parts, each machined from a different orientation.

In addition to the left-right division of bottom features, the beam is further segmented into head, middle, and tail sections to facilitate sequential machining from head to tail. The segmentation procedure is as follows (see Figure 14):

(1) The user selects reference point 1 for defining the head section. Based on this point, the system generates a reference plane DTM1 perpendicular to the *Y*-axis and extracts the head section accordingly.

(2) The user selects reference point #2 to define the tail section. The system creates a second reference plane DTM2, also perpendicular to the *Y*-axis, to separate the remaining portion into the middle and tail sections.

(3) The system then generates a third reference plane DTM3, perpendicular to the *X*-axis and located at the midpoint of the beam, to divide the beam into left and right halves.

The machining sequence of beam features is organized based on the principles of minimizing tool orientation changes and reducing tool travel distance. As the machine table operates along the *Y*-axis, the longitudinal direction of the beam, machining typically begins from the head section, which is closest to the origin, and proceeds sequentially through the middle and tail sections, as illustrated in Figure 15(*a*). A typical beam includes approximately 50 to 100 machining features. The machining sequence across the head, middle, and tail sections is arranged as follows:

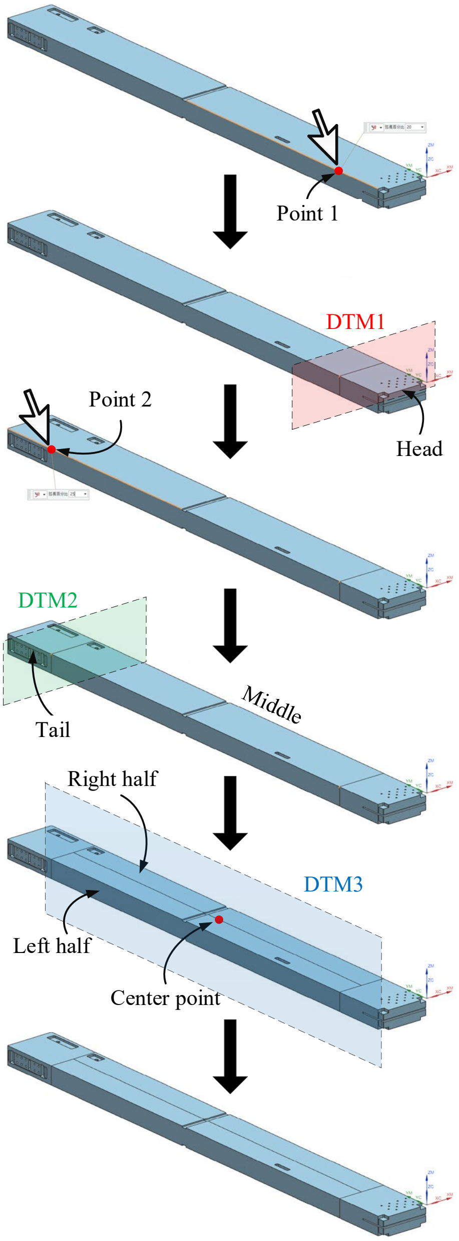


Fig. 14. Procedure for defining segmentation bodies

(1) Subdivision of sections: The head, middle, and tail sections are further divided into smaller subsections based on a predefined "section machining length." Machining begins with the subsection nearest to the origin and proceeds incrementally toward the tail. For example, in Figure 15(*b*), the machining order is as follows: (*a*) head section, (*b*) section #1, (*c*) section #2, (*d*) section #3, (*e*) section #4, (*f*) section #5, and (*g*) tail section. Sections #1 through #5 each have a default machining length of 1,000 mm, whereas the head and tail sections are shorter and thus not subdivided further.

(2) Machining sequence for general features: Within each subsection, the machining sequence for general features is arranged in two stages: bottom features are machined first, followed by non-

bottom features. For each type, the sequence is determined as follows:

(*a*) Start with the feature closest to the origin.

(*b*) Then machine the feature closest to the previously machined one.

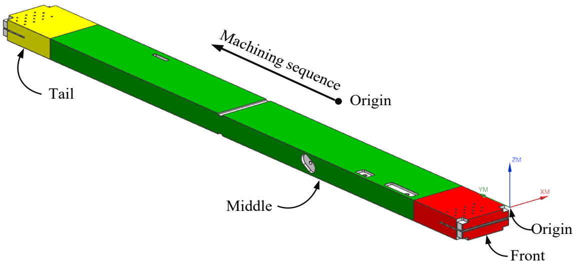
(*c*) Repeat step (*b*) until all features of that type are completed.

For instance, in Figure 15(*c*), section #1 contains four general features: features 1 and 2 are bottom features, while features 3 and 4 are non-bottom. The bottom feature closest to the origin is feature 1, followed by feature 2. Among non-bottom features, feature 4 is closer to the origin than feature 3. Therefore, the machining sequence for section #1 is: feature 1 → feature 2 → feature 4 → feature 3.

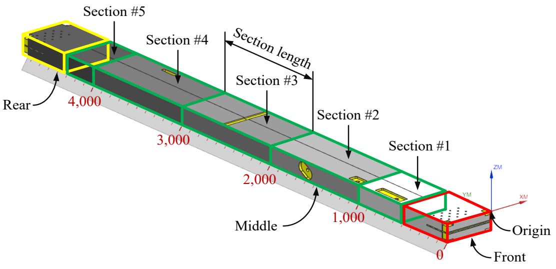
(3) Machining sequence for circular holes: For circular holes within each subsection, the same nearest-neighbor strategy is applied:

(*a*) Begin with the hole feature closest to the origin.

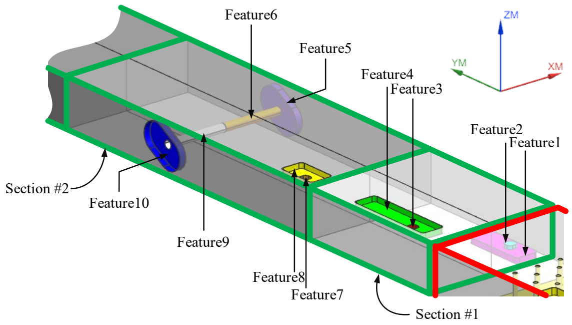
(*b*) Continue with the feature closest to the last machined one.



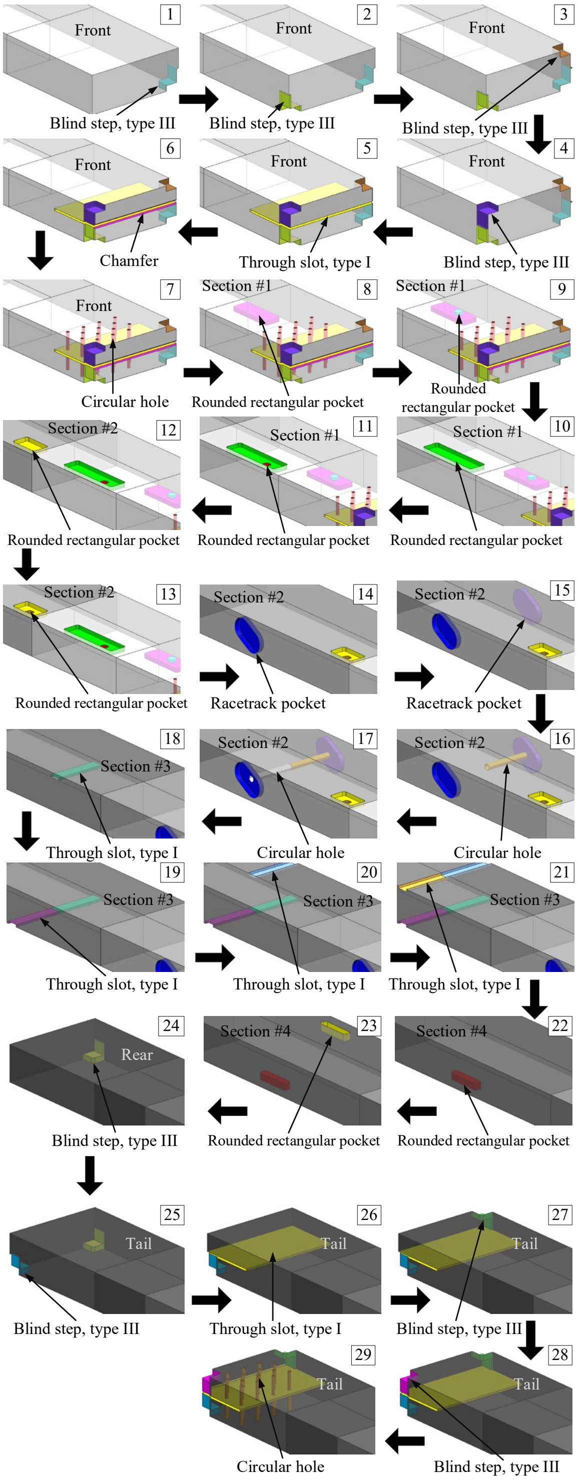
(*a*) Machining sequence for head, middle, and tail sections



(*b*) Example of machining sequence for subsections



(*c*) Features of subsections #1 and #2



(*d*) Example of machining sequence

Fig. 15. Machining sequence of wooden beam features

(*c*) Repeat step (*b*) until all hole features are completed.

In Figure 15(*c*), section #1 contains no hole features, so no hole machining is required.

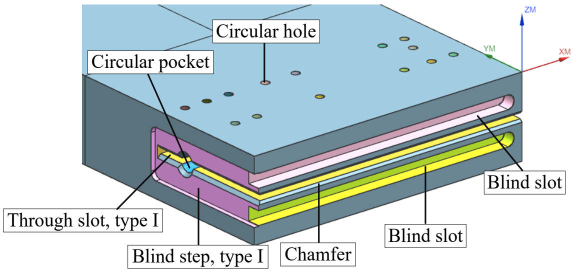
(4) Steps (2) and (3) are repeated for each subsection until the complete machining sequence for all features across the entire beam is determined. For

example, section #2 in Figure 15(*c*) includes four general features (features 5, 7, 8, and 10) and two hole features (features 6 and 9). Among the general features, feature 8 is closest to the origin, followed by features 7, 10, and 5. For hole features, feature 6 is closer to the origin than feature 9. Accordingly, the machining sequence for section #2 is: feature 8 → feature 7 → feature 10 → feature 5 → feature 6 → feature 9.

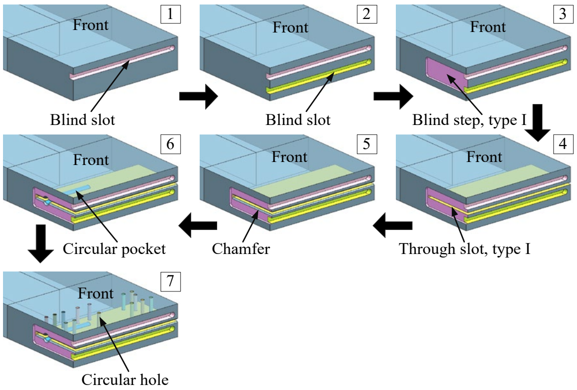
This sequencing strategy is consistently applied across all subsections, with the final comprehensive machining sequence summarized in Figure 15(*d*).

SYSTEM IMPLEMENTATION AND CASE STUDY

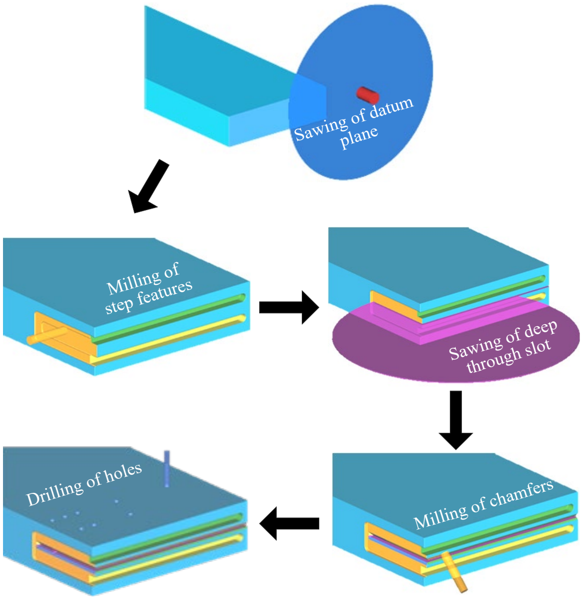
This study employs C++ in conjunction with Siemens NX's development platform, NX Open (2014), within the Visual Studio environment to develop a CAPP system for wooden beam machining. As shown in Figure 16(*a*), feature recognition identifies a total of 33 machining features in the head section of a beam, comprising 28 circular holes, 2 blind slots, 1 blind step (Type I), 1 circular pocket, and 1 through slot (Type I). Figure 16(*b*) presents the corresponding machining sequence, while Figure 16(*c*) illustrates the toolpath simulation. The machining process begins with sawing the reference surface, followed by cavity milling, saw cutting, and floor wall milling for the step features, deep through slots, and chamfer features, respectively. The process concludes with drilling the circular holes.



(*a*) Recognition results of features



(*b*) Sequence of feature machining



(*c*) Tool path simulation

Fig. 16. Machining of head section features

CONCLUSION

This study presents a fully automated process planning system for wooden beam machining that integrates feature recognition, process sequencing, tool orientation planning, and toolpath generation into a unified, intelligent framework. By leveraging face connectivity graphs and a novel matrix permutation and feature encoding method, the system accurately identifies not only standard machining features but also complex intersecting ones. This ensures precise, reliable classification and eliminates the ambiguities commonly encountered in graph-based feature recognition. The proposed approach also addresses real-world constraints in NC machining by segmenting beams into head, middle, and tail sections, adopting collision-free tool orientations, and arranging the machining sequence to minimize tool changes and travel distances. The system's capability to automatically convert 3D CAD models into complete machining programs with minimal human intervention marks a significant advancement toward CAD/CAM integration in beam processing.

ACKNOWLEDGMENT

Financial support for this work was provided by the National Science Council Taiwan, R.O.C, under the contract MOST 111-2221-E-011-145.

REFERENCES

Babić, B., Nešić, N. and Miljković, Z., "A review of automated feature recognition with rule-based pattern recognition," *Computers in Industry*, Vol.

59, No. 4, pp. 321-337 (2008).

Babić, B.R., Nešić, N. and Miljković, Z., "Automatic feature recognition using artificial neural networks to integrate design and manufacturing: Review of automatic feature recognition systems," *Artificial Intelligence for Engineering Design, Analysis and Manufacturing*, Vol. 25, No. 3, pp. 289-304 (2011).

Cao, P.X., "Key Machines Used in Light Frame Housing Construction," *China Wood Industry*, Vol. 24, No. 1, pp. 19-22 (2010).

Colligan, A.R., Robinson, T.T., Nolan, D.C., Hua, Y. and Cao, W., "Hierarchical CADNet: Learning from B-Reps for machining feature recognition," *Computer-Aided Design*, Vol. 147, 103226 (2022).

Ding, S., Guo, Z., Wang, B., Wang, H. and Ma, F., "MBD-Based Machining Feature Recognition and Process Route Optimization," *Machines*, Vol. 10, 906 (2022).

Han, J., Pratt, M. and Regli, W.C., "Manufacturing feature recognition from solid models: a status report," *IEEE Transactions on Robotics and Automation*, Vol. 16, No. 6, pp. 782-796 (2000).

Han, J.H., "Feature Recognition: the State of the Art," *Korean Journal of Computational Design and Engineering*, Vol. 3, No. 1, pp. 68-85 (1998).

Jones, T.J., Reidsema, C. and Smith, A., "Automated Feature Recognition System for supporting conceptual engineering design," *International Journal of Knowledge-Based and Intelligent Engineering Systems*, Vol. 10, No. 6, pp. 477-492 (2006).

Khan, M.T., Feng, W., Chen, L., Ng, Y.H., Tan, N.Y.J. and Moon, S.K. "Automatic feature recognition and dimensional attributes extraction from CAD models for hybrid additive-subtractive manufacturing,*" International Design Engineering Technical Conferences and Computers and Information in Engineering Conference*, August 25-28, Washington, DC (2024).

Lambourne, J.G., Willis, K.D., Jayaraman, P.K., Sanghi, A., Meltzer, P. and Shayani, H., "BRepNet: A topological message passing system for solid models," *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition* (Virtual Conference), June 19-25, pp. 12773-12782 (2021).

Li, W.D., Ong, S.K. and Nee, A.Y., "Recognizing manufacturing features from a design-by-feature model," *Computer-Aided Design*, Vol. 34, No. 11, pp. 849-868 (2002).

Liu, J., Liu, X., Cheng, Y. and Ni, Z., "An approach to mapping machining feature to manufacturing feature volume based on geometric reasoning for process planning," *Proceedings of the Institution of Mechanical Engineers, Part B: Journal of Engineering Manufacture*, Vol. 231, No. 7, pp. 1204-1216 (2017).

Manafi, D., Nategh, M.J. and Parvaz, H., "Extracting the manufacturing information of machining features for computer-aided process planning systems," *Proceedings of the Institution of Mechanical Engineers, Part B: Journal of Engineering Manufacture*, Vol. 231, No. 12, pp. 2072-2083 (2017).

Marchetta, M.G. and Forradellas, R.Q., "An artificial intelligence planning approach to manufacturing feature recognition," *Computer-Aided Design*, Vol. 42, No. 3, pp. 248-256 (2010).

Ning, F., Shi, Y., Cai, M. and Xu, W., "Part machining feature recognition based on a deep learning method," *Journal of Intelligent Manufacturing*, Vol. 34, pp. 809-821 (2023).

NX Open User Manual, Siemens PLM Software Inc., Plano, TX, USA (2014).

Shah, J.J., Anderson, D., Kim, Y.S. and Joshi, S., "A Discourse on Geometric Feature Recognition From CAD Models," *Journal of Computing and Information Science in Engineering*, Vol. 1, No. 1, pp. 41-51 (2000).

Shi, Y., Zhang, Y., Xia, K. and Harik, R., "A critical review of feature recognition techniques," *Computer-Aided Design and Applications*, Vol. 17, No. 5, pp. 861-899 (2020-1).

Shi, Y., Zhang, Y. and Harik, R., "Manufacturing feature recognition with a 2D convolutional neural network," *CIRP Journal of Manufacturing Science and Technology*, Vol. 30, pp. 36-57 (2020-2).

Shi, P., Qi, Q., Qin, Y., Scott, P.J. and Jiang, X., "A novel learning-based feature recognition method using multiple sectional view representation," *Journal of Intelligent Manufacturing*, Vol. 31, pp. 1201-1309 (2020-3).

Verma, A.K. and Rajotia, S., "A hint-based machining feature recognition system for 2.5 D parts," *International Journal of Production Research*, Vol. 46, No. 6, pp. 1515-1537 (2008).

Verma, A.K. and Rajotia, S., "A review of machining feature recognition methodologies," *International Journal of Computer Integrated Manufacturing*, Vol. 23, No. 4, pp. 353-368 (2010).

Watanabe, Y. and Nakamoto, K., "Proposal of a machining features recognition method for 5-axis index milling on multi-tasking machine tools," *Journal of Advanced Mechanical Design Systems and Manufacturing*, Vol. 14, No. 7, pp. 1-13 (2020).

Wu, H., Lei, R., Peng, Y. and Gao, L., "AAGNet: A graph neural network towards multi-task machining feature recognition using geometric Attributed Adjacency Graph representation," *Robotics and Computer-Integrated Manufacturing*, Vol. 86, 102661 (2024).

Yang, J., Wu, Q., Zhang, Y., Dai, J. and Wang, J., "A hybrid recognition framework for highly interacting machining features based on primitive decomposition, learning and reconstruction," *Computer-Aided Design*, Vol. 179, 103813 (2025).

Zhao, Y., Yang, C.M., Qi, Y.J. and Yang, C.Z., "Development of Wood Architecture Since the

Founding of New China," Forestry Machinery & Woodworking Equipment, Vol. 40, No. 5, pp. 10-16 (2012).