Front. Electr. Electron. Eng. China 2008, 3(4): 440–458
DOI 10.1007/s11460-008-0071-x

RESEARCH ARTICLE

Hao DAI, Xiaojun SHEN

Rearrangeability of 7-stage 16 × 16 shuffle exchange networks

Abstract It has long been an outstanding conjecture that any $(2n-1)$-stage shuffle exchange network (Omega network) is rearrangeable for $2^n \times 2^n$. Many researchers have failed to prove this conjecture, including a recent one established by Hasan. However, nobody has pointed out its fallacy. Therefore, as one of the objectives, this paper shall clarify this fact. Since the case of $n = 3$ has been proven by many researchers [1,2], this paper uses a constructive approach to prove that when $n = 4$, the 7-stage 16×16 shuffle exchange network is also rearrangeable. The paper also presents the model of a balanced tree to avoid internal conflict, the representation of permutations using a connection graph and loop graph, and the concepts of symmetry graph and identical transform. Based on graphic composition and bipartition, the permutations 16×16 are divided into five classes, with five assignment algorithms proposed. These algorithms are simpler, clearer and easier to program. The techniques used for $n = 4$ may provide hints for the general case of $n > 4$.

Keywords multistage interconnection network (MIN), shuffle exchange network, rearrangeability, equivalent transform, identical transform

1 Introduction

The shuffle exchange network offered by H. S. Stone in 1971 is used on large-scale parallel computing systems and telecommunication switches [3]. It usually has N inputs and N outputs (note $N \times N$, $N = 2^n$) and is called multistage interconnection network (MIN), which consists of at least n stages (stairs), and where each stage consists of $N/2$ 2×2 switching elements (SEs). An SE has two inputs and two outputs and can only be set to the through or cross state. The inter-stage connections that realize the characters are called a perfect shuffle. The Omega network (denoted by Ω) is the typical representation of a perfect shuffle exchange network [4]. In the Ω with n stages, a pair of input and output addresses has a unique path, as well as self-routing capability, i.e., according to binary source address $S_1S_2S_3S_4$ and binary destination address $D_1D_2D_3D_4$, we can decide the states of each SE along the path. Figure 1 shows a 16×16 Omega network with 4 stages.

The N-source address and N-destination address in the MIN exhibit a one-to-one correspondence, equaling a permutation with N natural numbers. There are $N!$ permutations for natural number N. After eliminating output conflict, there may still be internal blocking or conflict when the N-input across network aims to reach the N-output at one time. Some permutations such as ideal or line permutations can find the N-path without any conflict in an Ω network [5], but the total number of these permutations is a small portion of $N!$. A multistage shuffle exchange network is called rearrangeable or permutationable if it is admissible for $N!$ permutations without internal conflict.

In an n-stage Ω network, there are two methods of realizing rearrangeability. One is to partition the N-permutation into several groups, each of which can pass through the Ω network at one time. This method is characterized by seeking the minimum total number of groups [6]. The other approach is to increase the stage number to pass through the Ω network without blocking at one time for all N cells that move to a distinct output. Because the path across an extended network for a given pair of input and output is not single, more n-stage Ω networks have the capability of avoiding internal blocking, which is usually difficult to avoid in an n-stage Ω network. This method is characterized by seeking the stage's lower bound.

Translated from *Acta Electronica Sinica*, 2007, 35(10): 1875–1885 [译自: 电子学报]

Hao DAI (✉)
Institute of China Electronic Equipment System Engineering Company, Beijing 100036, China
E-mail: dai_hao@sina.com

Xiaojun SHEN
School of Computing and Engineering, University of Missouri-Kansas City, Kansas City, MO 64110, USA

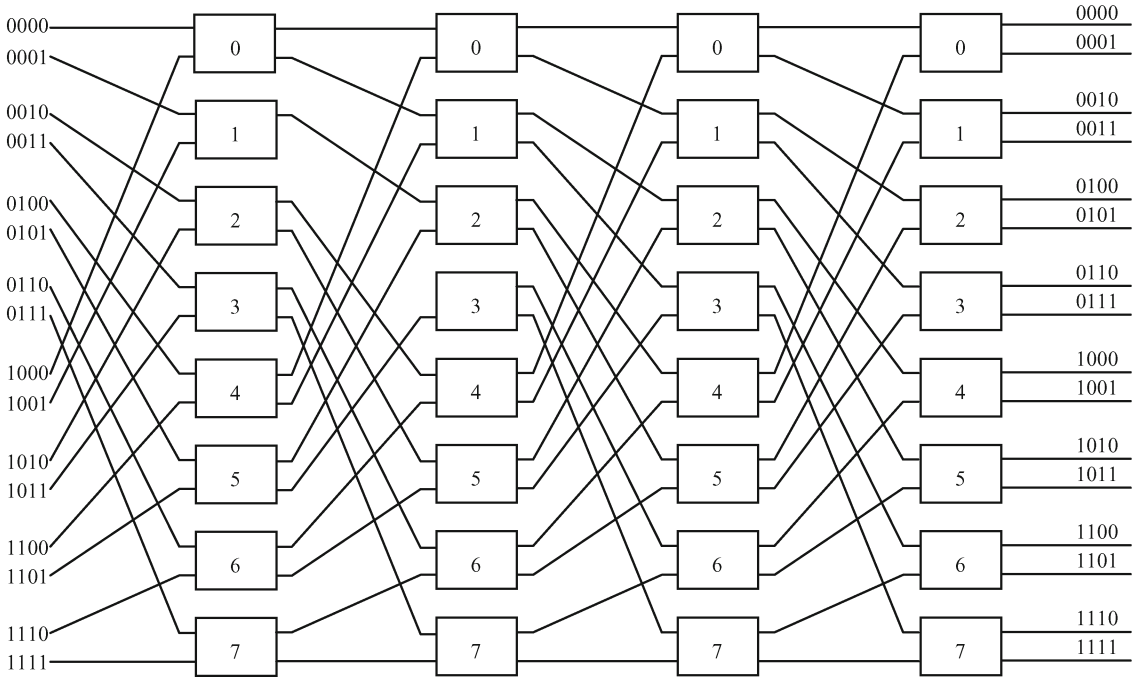


Fig. 1 Topological representation of Ω network

Due to progressing integrated circuit technology and dropping electronic parts costs, researchers have given more attention to the second method. For 30 years, many scholars have been devoted to this fundamental problem [7–11]. In 1975, Benes conjectured that the $(2n-1)$-stage was the sufficient and necessary condition for the rearrangeability of a shuffle exchange network Ω [12]. To prove the sufficiency of the conjecture is easy, but to prove the necessity is very difficult. With an expend loop algorithm, it was proven that Benes' network with $(2n-1)$-stage is rearrangeable [13], but whether it is the sufficiency condition to realize rearrangeability of the Omega network is still an issue. Many researchers have attempted to prove this conjecture with Boolean algebra, finite-state machine model, graph theory, matrix methods and acquired lower bounds such as $2n^2$, $\left(3n^2-n\right)/2$, n^2, $3n$, $3n-1$, $3n-3$, $3n-4$ [3,7,9,14–17]. However, no one has ever reached the lowest bound of $(2n-1)$. Several articles apply topological equivalence in proving that all types of shuffle exchange networks were isomorphic [18–20], which was later proven incorrect [5,21]. Recently, Hasan proposed a sufficiency proof about the rearrangeability of the $(2n-1)$-stage Ω network [22]. He constructed a set of balanced matrixes with $N \times (2n-1)$ stages, from which he filtered out the proper matrix. However, he neither proved that for any permutation, balanced matrixes are always non-empty after a series screening, nor did he give an example of the constructive approach. Considering the need of engineering application, a simple and efficient routing algorithm is required.

Currently, only the constructive algorithm for $n=3$ 5-stage Ω network rearrangeability has been realized in 1987 [1,2,16], thereby proving the sufficiency of the conjecture. Thereafter, no breakthrough was achieved in this field. In Ref. [5], it is stated that the conjecture had been verified in case $n=4$ by running a program that exhaustively checks all permutations. However, no analytical proof has been reported until now even in case $n=4$. Efforts were made to classify various samples to solve the problem, but there are too many cases to deal with and the algorithms are also very tricky. This paper offers an analytical algorithm for $n=4$ 7-stage Ω network rearrangeability. Using graphic representation of permutation and principle of equivalent transform, we classified 16! permutations into 5 cases with corresponding routing algorithms for each case.

The rest of this paper is organized as follows: Section 2 introduces background knowledge and basic concepts such as matrix representation of permutation, concept of transfer matrix, a window check method for internal conflict and the model of the balanced tree. Section 3 presents the graphic representation of permutation, the composition and bipartition of a connection graph, equivalent transform and the classification of the permutations. Section 4 introduces the symmetry of permutation and two basic assignment algorithms. Section 5 is the key to this paper, presenting identical transform, loop graph representation of permutation and three assignment algorithms based on identical transforms. Finally, this paper concludes with the bipartition principles for connection or loop graphs, and discusses rearrangeability for a multistage Ω network when $n>4$.

2 Preliminary knowledge

2.1 Matrix representation of permutation

In this paper, a source address $S = S_1S_2S_3S_4$ and a destination address $D = D_1D_2D_3D_4$ are denoted by 4 binary digits. Combining the vector of source and destination address, the corresponding permutation can be denoted by a 16×8 matrix $\boldsymbol{P} = [S_1S_2S_3S_4D_1D_2D_3D_4]$. Since every entry is either 0 or 1, it is also called an 0–1 matrix. The rows of matrices are numbered in ascending order, from 0 up to 15 and from top to bottom. In the matrix $\boldsymbol{P}$, any two rows can be interchanged by reciprocating their positions since they express the same permutation. The following equation shows two matrix forms that correspond to a particular permutation, where in the first matrix the left source address is arranged in converse order, i.e., $S_4S_3S_2S_1$ is in ascending order from 0000, 0001, 0010, …, up to 1111, while in the second matrix the right destination address is arranged in normal order, i.e., $D_1D_2D_3D_4$ is in ascending order from 0000, 0001, 0010, …, up to 1111.

$$\boldsymbol{P} = \begin{array}{c} \begin{matrix} S_1 & S_2 & S_3 & S_4 & D_1 & D_2 & D_3 & D_4 \end{matrix} \\ \begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 \\ 1 & 0 & 0 & 0 & 0 & 0 & 1 & 1 \\ 0 & 1 & 0 & 0 & 0 & 1 & 0 & 1 \\ 1 & 1 & 0 & 0 & 1 & 1 & 1 & 1 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 \\ 1 & 0 & 1 & 0 & 1 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 & 1 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 & 0 \\ 1 & 0 & 0 & 1 & 0 & 1 & 1 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 1 & 1 & 1 & 0 & 1 \\ 0 & 0 & 1 & 1 & 1 & 0 & 0 & 0 \\ 1 & 0 & 1 & 1 & 1 & 0 & 1 & 1 \\ 0 & 1 & 1 & 1 & 1 & 0 & 1 & 0 \\ 1 & 1 & 1 & 1 & 1 & 0 & 0 & 1 \end{pmatrix} \end{array} = \begin{array}{c} \begin{matrix} S_1 & S_2 & S_3 & S_4 & D_1 & D_2 & D_3 & D_4 \end{matrix} \\ \begin{pmatrix} 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 0 & 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 1 & 1 \\ 0 & 1 & 1 & 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 1 & 0 & 1 \\ 1 & 0 & 0 & 1 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 & 1 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 & 1 & 0 & 0 & 1 \\ 0 & 1 & 1 & 1 & 1 & 0 & 1 & 0 \\ 1 & 0 & 1 & 1 & 1 & 0 & 1 & 1 \\ 1 & 0 & 1 & 0 & 1 & 1 & 0 & 0 \\ 1 & 1 & 0 & 1 & 1 & 1 & 0 & 1 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 & 0 \\ 1 & 1 & 0 & 0 & 1 & 1 & 1 & 1 \end{pmatrix} \end{array}. \tag{1}$$

2.2 Checking internal conflict

Suppose that in an Ω network, each stage numbered in ascending order starting with 1 from left to right (Fig. 2), and in a 4-stage Ω network the self-routing algorithm decides the state of each stage. Given a permutation composed of a group of source and destination addresses, the state of switch on i-stage is decided by the value $S_i \oplus D_i$. When $S_i = D_i$, the switch state is set through, otherwise, the switch state must be set across. In other words, the bit value of the destination address D_i decides a switch state or output. Generally, $D_i = 0$ means the path outputs from its upper end, while $D_i = 1$ means the path outputs from its lower end. The internal conflict means two paths entering a switch from different ports while being requested to exit via the same port, or entering a switch from the same port while exiting via different ports. They require a switch to maintain through and cross states at the same time, but this is impossible. Figure 2 shows an example of an Ω network with two paths: $(0010 \rightarrow 0111)$ and $(1010 \rightarrow 0001)$. After passing over a switch in stage 1, they all have to exit from the address 0100 to enter the next state. Obviously, these two paths cannot be established simultaneously.

To check internal conflict for more paths in an Ω network, a window checking method is usually introduced. In a 16×8 permutation matrix $\boldsymbol{P}$, each 4 consecutive columns constitute a sub-matrix, which is also viewed as a series window. If a window consists of 16 different 4-bit binary numbers, i.e., there is a full array from 0000 to 1111, then corresponding switches on the same stage will not produce conflict because they occupy different output addresses. In the permutation matrix $\boldsymbol{P}$, the first and the final window are input and output address matrices, with different binary strings. Hence, we need only check 3 windows in the middle of $\boldsymbol{P}$. Generally, $\boldsymbol{P}$ is composed of a source and destination address, which can hardly satisfy the full array condition.

2.3 Transform matrix of permutation

If a stage is added to a 4-stage Ω network, for a particular connection, the available paths will be doubly increased; in adding two stages, the availability will increase fourfold; for three stages, it will be eightfold. Therefore, an expended

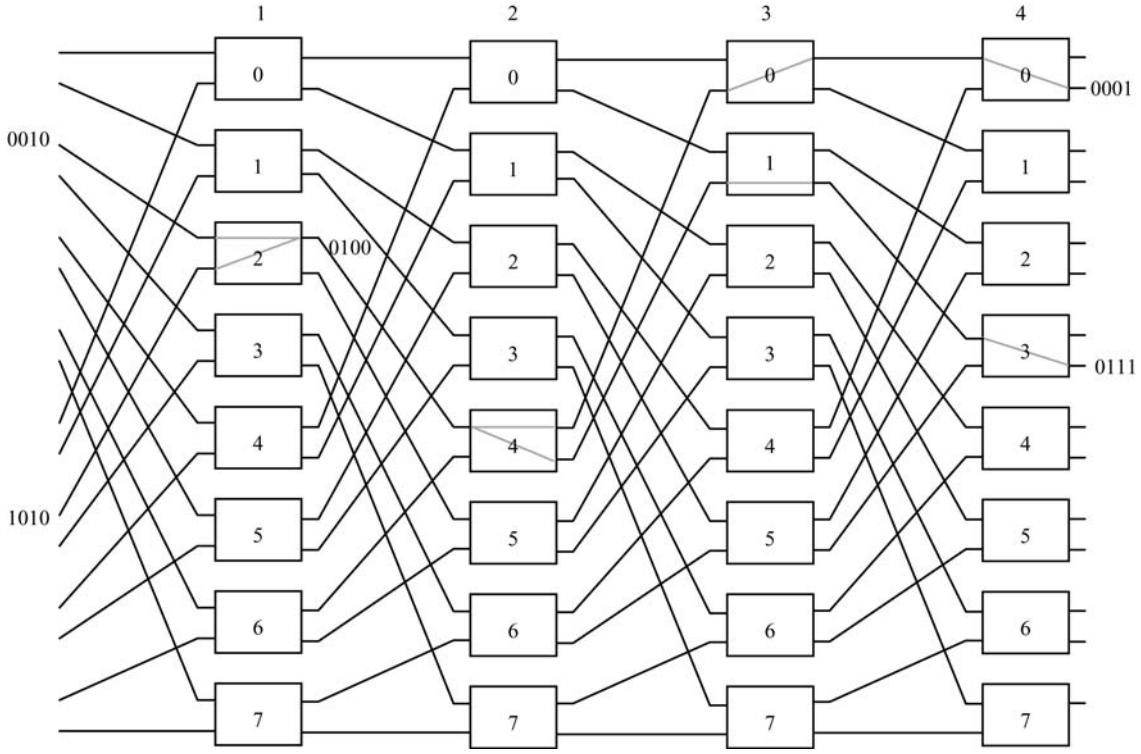


Fig. 2 Internal conflict of two paths in 4-stage Ω network

7-stage Ω network may avoid internal conflict and allow 16 routes to pass the network at one time. Because there are a total of $8 \times 7 = 56$ switches and every switch has two states, the total solution space measures about $2^{56} = 6.4 \times 10^{16}$, more than the problem space that measures about $16! = 2.1 \times 10^{13}$. The key problem is how to find a certain and conflict-free route for given connection, which means searching the output address for each switch along a path. Therefore, we need to expand the permutation matrix $\boldsymbol{P}$[16,8] into the transfer matrix $\boldsymbol{T}$[16,11], i.e., add another 3 columns in the middle of a matrix $\boldsymbol{P}$, denoted by $[XYZ]$, so that

$$\boldsymbol{T} = [S_1S_2S_3S_4XYZD_1D_2D_3D_4].$$

If $\boldsymbol{T}$[16,11] passes window checking, it means any sub-matrix (window) constituted by 4 consecutive columns has different binary strings, and the 16 routes must be conflict-free in passing a 7-stage Ω network [16]. Therefore, the rearrangeability problem turns into filling in $\boldsymbol{T}$ with a 3-bit string for each column of XYZ. We need to find 16 binary numbers, each occurring exactly twice. For a particular connection, the three bits are added to the front of the destination address string, which is also called the routing prefix or expended output address. All switch states along the path are determined by the value $\boldsymbol{T}_{i,j} \oplus \boldsymbol{T}_{i,j+4}$ $(0 \leqslant i \leqslant 15, 1 \leqslant j \leqslant 7)$.

2.4 Balanced tree model of non-conflict

To assure six windows $[S_2S_3S_4X]$, $[S_3S_4XY]$, $[S_4XYZ]$, $[XYZD_1]$, $[YZD_1D_2]$ and $[ZD_1D_2D_3]$ in the transfer matrix $\boldsymbol{T}$ to be fully arranged, i.e., the binary number in every row is different, we propose a concept of multi-dimensional address space. It requires three columns XYZ to satisfy the following conditions.

Left balance in 1-dimensional space: when $S_1S_2S_3$ is the same, the corresponding X is different.

Left balance in 2-dimensional space: when S_1S_2 is the same, the corresponding XY is different.

Left balance in 3-dimensional space: when S_1 is the same, the corresponding XYZ is different.

Right balance in 1-dimensional space: when $D_1D_2D_3$ is the same, the corresponding Z is different.

Right balance in 2-dimensional space: when D_1D_2 is the same, the corresponding YZ is different.

Right balance in 3-dimensional space: when D_1 is the same, the corresponding XYZ is different.

A XYZ bit value that satisfies the above conditions is called a completely valid assignment. When a matrix $\boldsymbol{T}$ is evenly divided into four quadrants, each quadrant represents a 3-dimensional address space. The six conditions listed above could be denoted by a balanced tree model. In Figs. 3 and 4, the two trees respectively present balanced conditions for quadrant II ($S_4 = 0$) and quadrant I ($D_1 = 0$), and every tree has three layers. Another two balanced trees have a similar structure.

The shuffle exchange network offers many solutions for one problem. For example, if vectors composed of XYZ bit values satisfy the request of the balanced tree, then it is completely valid to negate these values independently or

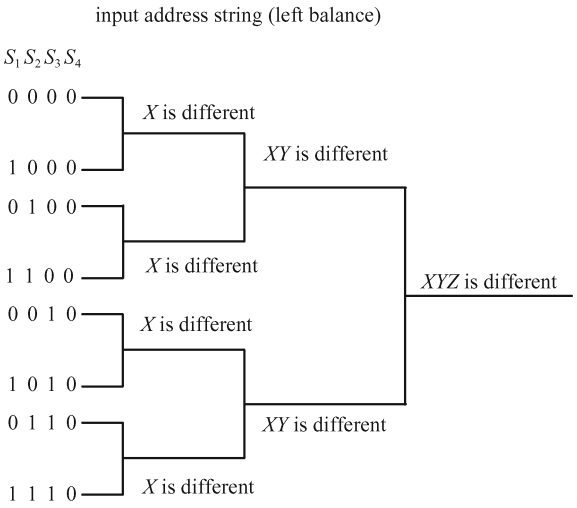


Fig. 3 Model of conflict-free tree for quadrant II

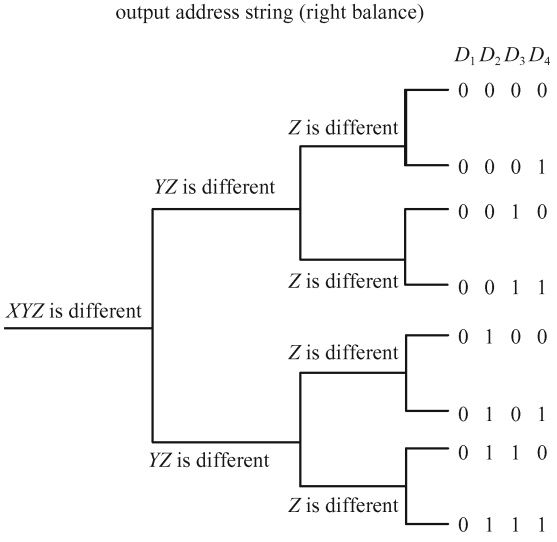


Fig. 4 Model of conflict-free tree for quadrant I

simultaneously, as well as forming a new group of vectors such as $\bar{X}YZ$, $X\bar{Y}Z$, $\bar{X}\bar{Y}Z$, $XY\bar{Z}$, $X\bar{Y}\bar{Z}$, $\bar{X}Y\bar{Z}$, $\bar{X}\bar{Y}\bar{Z}$

3 Decomposing connection graphs and classifying permutations

3.1 Graphic representation of permutation

Any 16 × 16 permutation can be represented by normal biography (even graphs), where the left vertices are denoted by an input binary address, and the right vertices by an output binary address. A line (edge) that connects left and right vertices denotes a permutation. On every edge a corresponding and reasonable XYZ bit value is assigned. Figure 5(a) draws two biographies that correspond to the permutation shown in Eq. (1).

Compressing biography propriety, two vertices that belong to the same 1-dimensional space (same $S_2S_3S_4$ or same $D_1D_2D_3$) are integrated into one. This forms a new connection graph which has 8 vertices on both left and right, as shown in Fig. 5(b) and denoted by G or $G(16)$. Each vertex in G has rank (degree) 2. Thus, two ends of adjacent edges are converged on a vertex, which means that they belong to the same 1-dimensional space. Two vertices belonging to the same 2-dimensional space (S_3S_4 is the same or D_1D_2 is the same) keep neighborhood relation. G and biography express a particular permutation $\boldsymbol{P}$. In summary, G is called the connection graph for permutation $\boldsymbol{P}$ and we can bipartite G into four quadrants using Descartes coordinate axes.

Definition 1 A cross edge is the edge whose input address belongs to quadrant III (II) and output address belongs to quadrant I (IV). A non-cross edge is the edge whose input address belongs to quadrant II (III) and output address belongs to quadrant I (IV).

Corollary 1 The cross edge number of quadrant III/I is the same as that of quadrant II/IV.

Corollary 2 In G the pair number of cross edge $m \leqslant 8$.

3.2 Equivalent transform for connection graph

In addition to the multi-solution for one problem, the shuffle exchange network is also characterized by one solution for multi-problems. For example, using the following Lemmas 1–4 to exchange and combine the input/output addresses will yield some new permutations that can be classified since they have the same answer. Under a topologically isomorphic precondition, some equivalent transforms of a connection graph are listed as follows:

Lemma 1 In the connection graph, two input/output addresses in a vertex (belonging to the same 1-dimensional space) can exchange their positions.

Lemma 2 In the connection graph, two groups of input/output addresses composed of two 1-dimensional spaces and reside in the same 2-dimensional space can exchange their positions.

Lemma 3 In the connection graph, two groups of input/output addresses that consist of two 2-dimensional spaces and reside in the same 3-dimensional space can exchange their positions.

Lemma 4 In the connection graph, two groups of input/output addresses that consist of an upper and lower quadrant (in the same 3-dimensionsl space) and reside on the same side of the graph can exchange their positions.

According to Lemma 4, after executing an equivalent transform for the connection graph, the cross edge and non-cross edge can exchange positions. If we suppose that there are m pairs of cross edges in G before exchanging, there will be $8-m$ pairs of cross edges after the change. Without loss of generality, in the following context we assume $m \leqslant 4$.

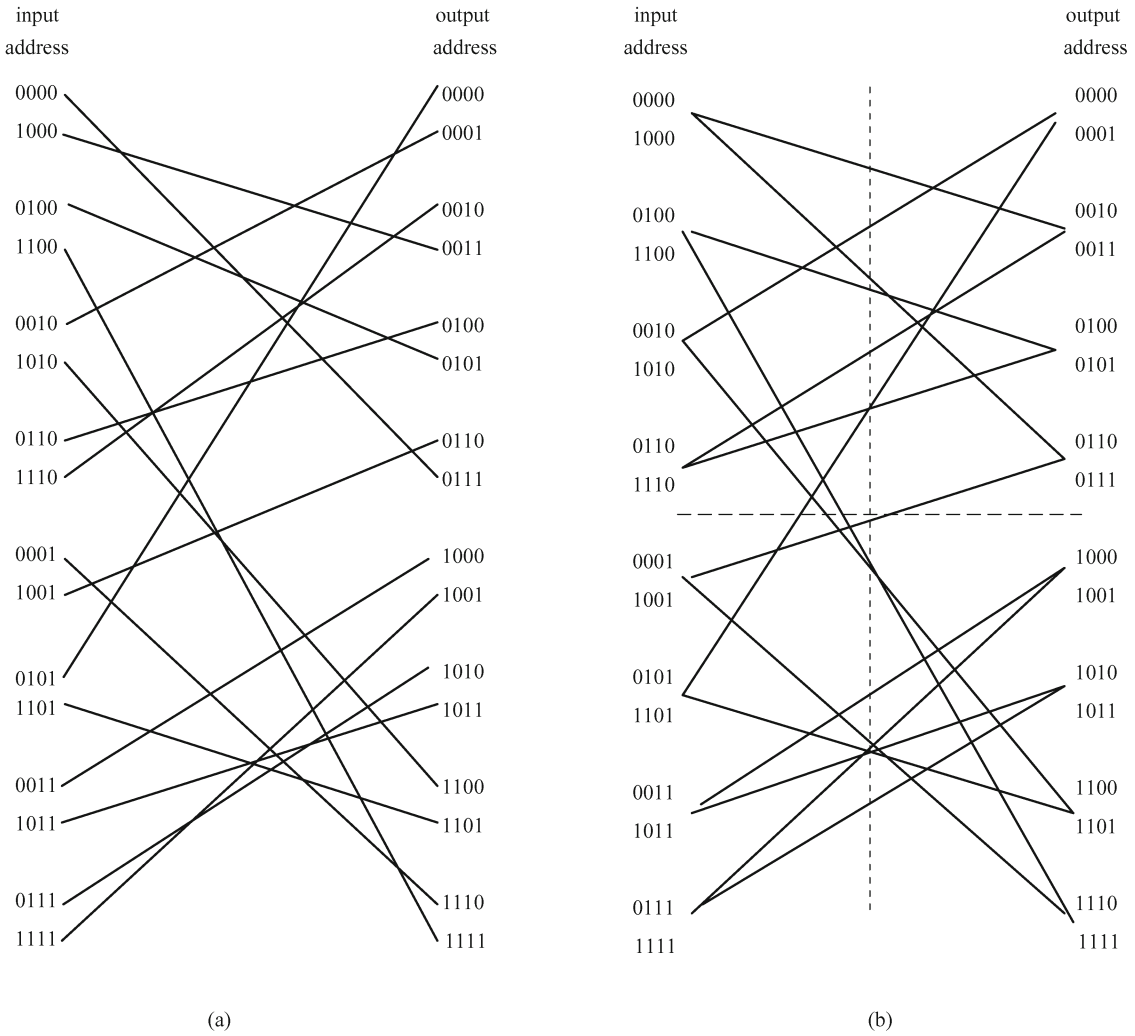


Fig. 5 (a) Permutation corresponding bi-graph; (b) compressed connection graph

Lemma 5 In a connection graph, after exchanging left input addresses and right output addresses the new permutation is a reverse of the original permutation.

Based on the equivalent transform above, after properly adjusting output address positions in a connection graph as shown in Fig. 5(b) and compressing the graph again, the two vertices in the same 2-dimensional space will be incorporated. We can then get regular connection graph G, as shown in Fig. 6. Following Lemmas 1–4, the vertex in the connection graph will no longer be labeled by binary address. We focus on how to properly assign XYZ bit values for the 16 edges of pure G.

3.3 Decomposing connection graphs

In a connection graph the 16 edges could form one cycle or more with different lengths. Taking alternate edges on the cycle, graph $G(16)$ will be divided into two parts, and converging two edge ends which are located in the same 2-dimensional address space, denoted by G_1 and G_2. In G_1 and G_2, taking alternate edges on the cycle again. Thus, G_1 is divided into two 4×4 biographies, denoted by G_{11} and G_{12}. Likewise, G_2 is divided into two 4×4 biographies, denoted by G_{21} and G_{22}. In each biography there are four edges, called four-tuple arrays, denoted by G_{ij} (i, $j = 1,2$). In Fig. 6, there are two methods of decomposing G, the corresponding sub-graph G_1 and G_2, and the four-tuple arrays are shown in Fig. 7.

Corollary 3 Suppose the length of a cycle is L. When G (G_i) has only one cycle, the bipartition into G_i (G_{ij}) is unique; when G (G_i) has two or more cycles with length $L \geqslant 4$, the bipartition into G_i (G_{ij}) is not unique.

Corollary 4 In a four-tuple G_{ij}, the cross edges are always in pairs. In G_{ij}, the number of cross edge pairs is two or less.

In theory there is a total of $4! = 24$ different topological structures G_{ij}, and the 13 typical structures are shown in Fig. 8, where 7 four-tuples have one pair of cross edges. For the four-tuple with zero or two pairs of cross edges, the assignment method is the same. Therefore, we focus on the four-tuples with one pair of cross edges. In the

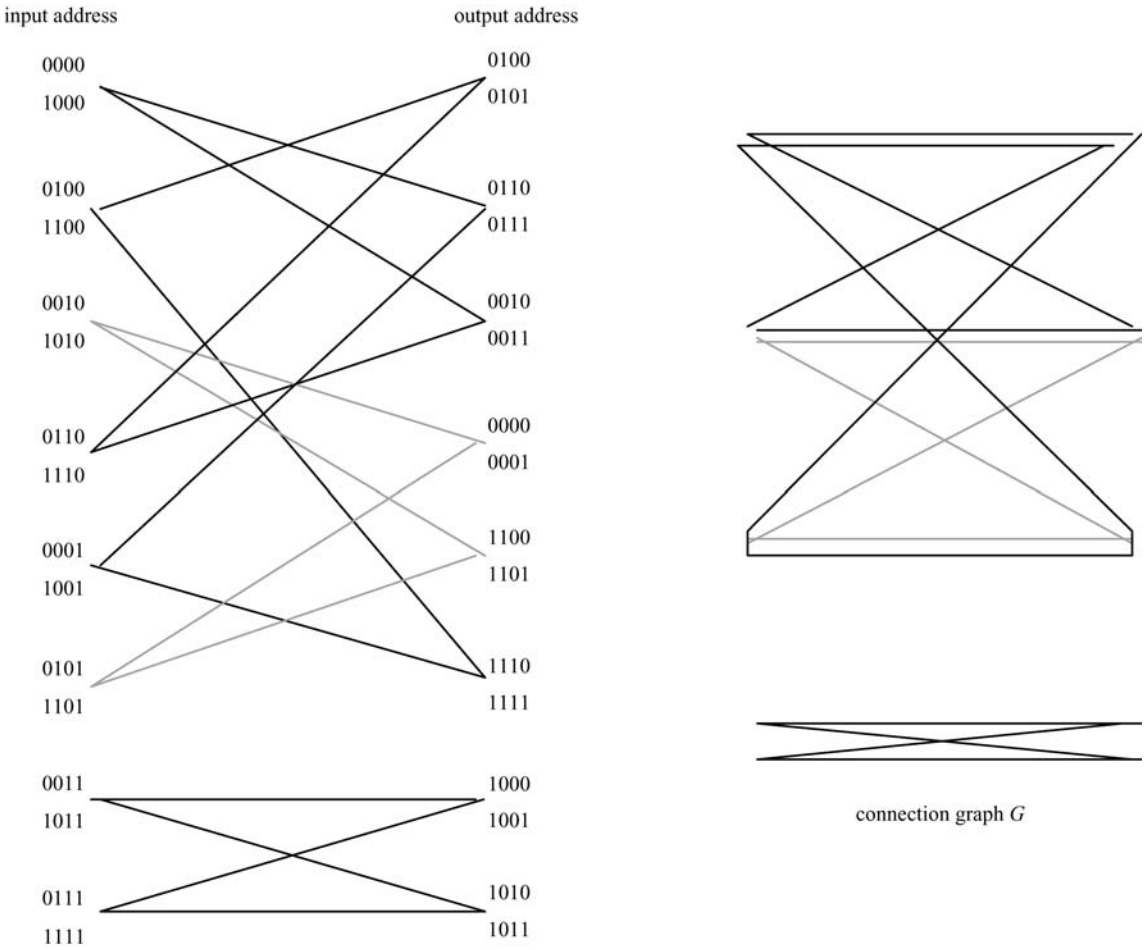


Fig. 6 Adjusted connection graph and twain compressed graph

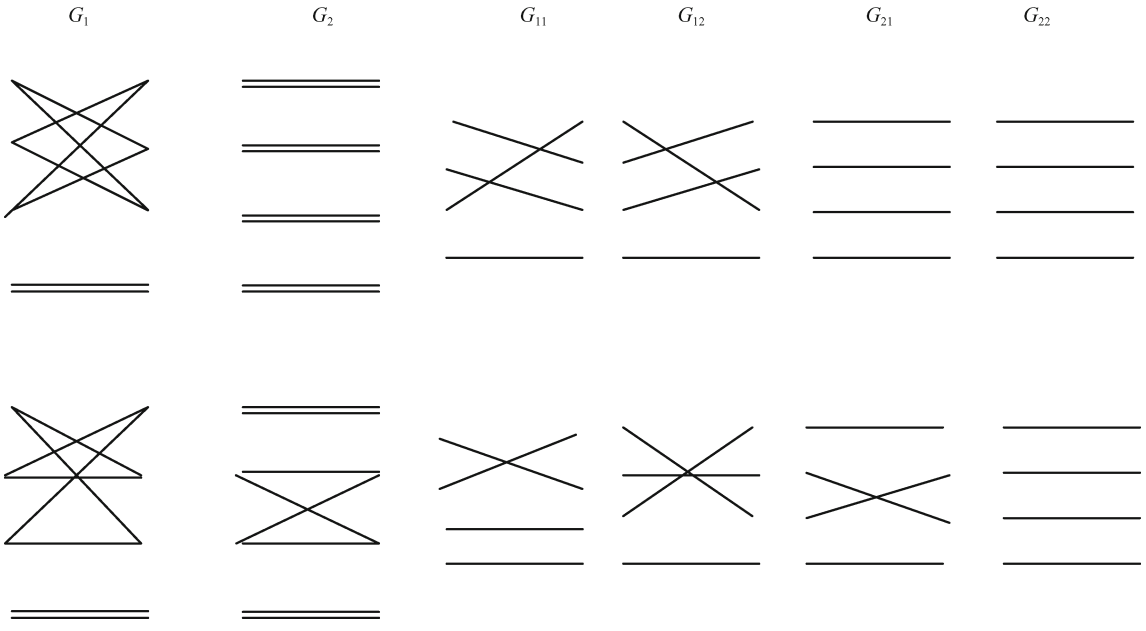


Fig. 7 Two bipartitions for connection graph G

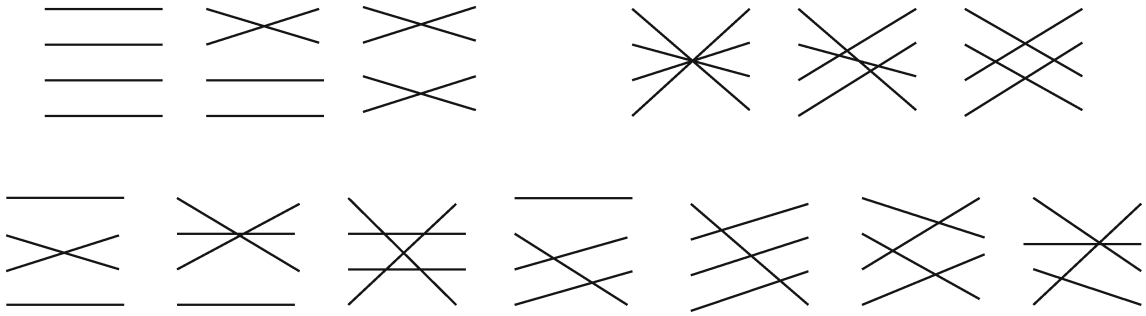


Fig. 8 Different topologic of G_{ij}

following, unless otherwise specifically stated, the G_{ij} with cross edges always indicates a G_{ij} with one pair.

3.4 Classifying permutations

After decomposing graph G into four G_{ij}, according to the cross edge number K, we can classify G into 5 cases.

Case 0 Neither G_{ij} with cross edges, i.e., $K = 0$;
Case 1 There is one G_{ij} with cross edges, i.e., $K = 1$;
Case 2 There are two G_{ij} with cross edges, i.e., $K = 2$;
Case 3 There are three G_{ij} with cross edges, i.e., $K = 3$;
Case 4 All G_{ij} with cross edges, i.e., $K = 4$.

Note This classification differs with the pairs m of cross edges in G. Table 1 lists their corresponding relations. In this table the symbol '=', '×', and '#' represent an G_{ij} with 0, 1, and 2 pair of cross edges respectively.

Table 1 Corresponding relations between m and five cases

cross edge pair m	Case 0	Case 1	Case 2	Case 3	Case 4
0	= = = =				
1		× = = =			
2	= = = #		× × = =		
3		× = = #		× × × =	
4	= = # #		× × = #		× × × ×

In the next section it will be explained that we can directly execute Algorithm 1 for assignment in Cases 0 and 1 and directly execute Algorithm 2 for assignment in Case 4. In Cases 2 and 3, when graph G is symmetric, we can directly execute Algorithm 1 for assignment. Otherwise, the end position needs to be properly adjusted to a cross edge and/or non-cross edge, i.e., carry out some identical transforms, and then execute Algorithm 1 or 2 for assignment. Thus, Algorithms 1 and 2 are basic.

4 Basic assignment algorithms

4.1 Symmetry of permutation

To introduce the concept of symmetric permutation or symmetric graph G, we divide each quadrant of G into upper and lower areas (each area contains all edges and their terminals have the same 2-dimensional address space), which are denoted by A, B, C, and D from top to bottom. In the left and right quadrants the label of the area is symmetric, as shown in Fig. 9. In G_i, G_{ij} (i, $j = 1, 2$) the label of the area is similar to that in G.

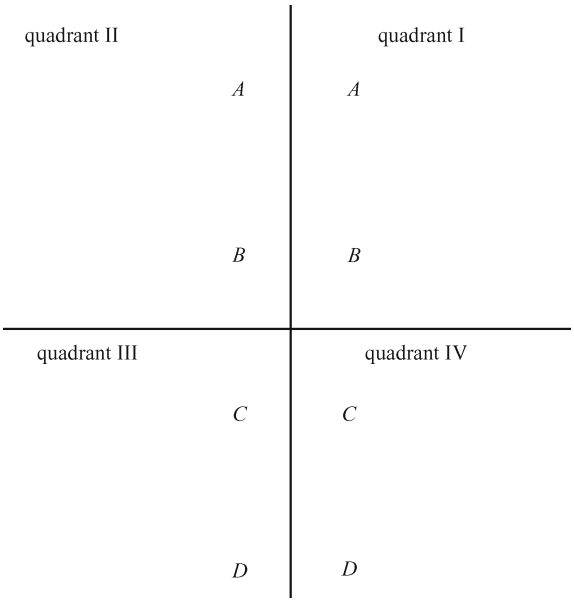


Fig. 9 Labels of 2-dimensional address space

Definition 2 Symmetry of graph

In Cases 2 and 3, if the right (left) terminal of each pair of cross edges in G_{ij} respectively locates in area A, D and/or area B, C (see Fig. 10(a)), or respectively locates in area A, C and/or area B, D (see Fig. 10(b)), then these terminals of cross edges in G are called right (left) symmetric. When the cross edges distribution is right and/or left symmetric, then G is also symmetric. Otherwise, G and its cross edges are asymmetric, i.e., it is neither right symmetric nor left symmetric (see Fig. 10(c)).

In Case 1, G has a native symmetric character. In Cases 0 and 4, the concept of symmetric G is not applicable.

Based on Lemmas 2–5, we could make G standardized. Without loss of generality, we suppose that a symmetric graph is always right symmetric, and the ends of cross

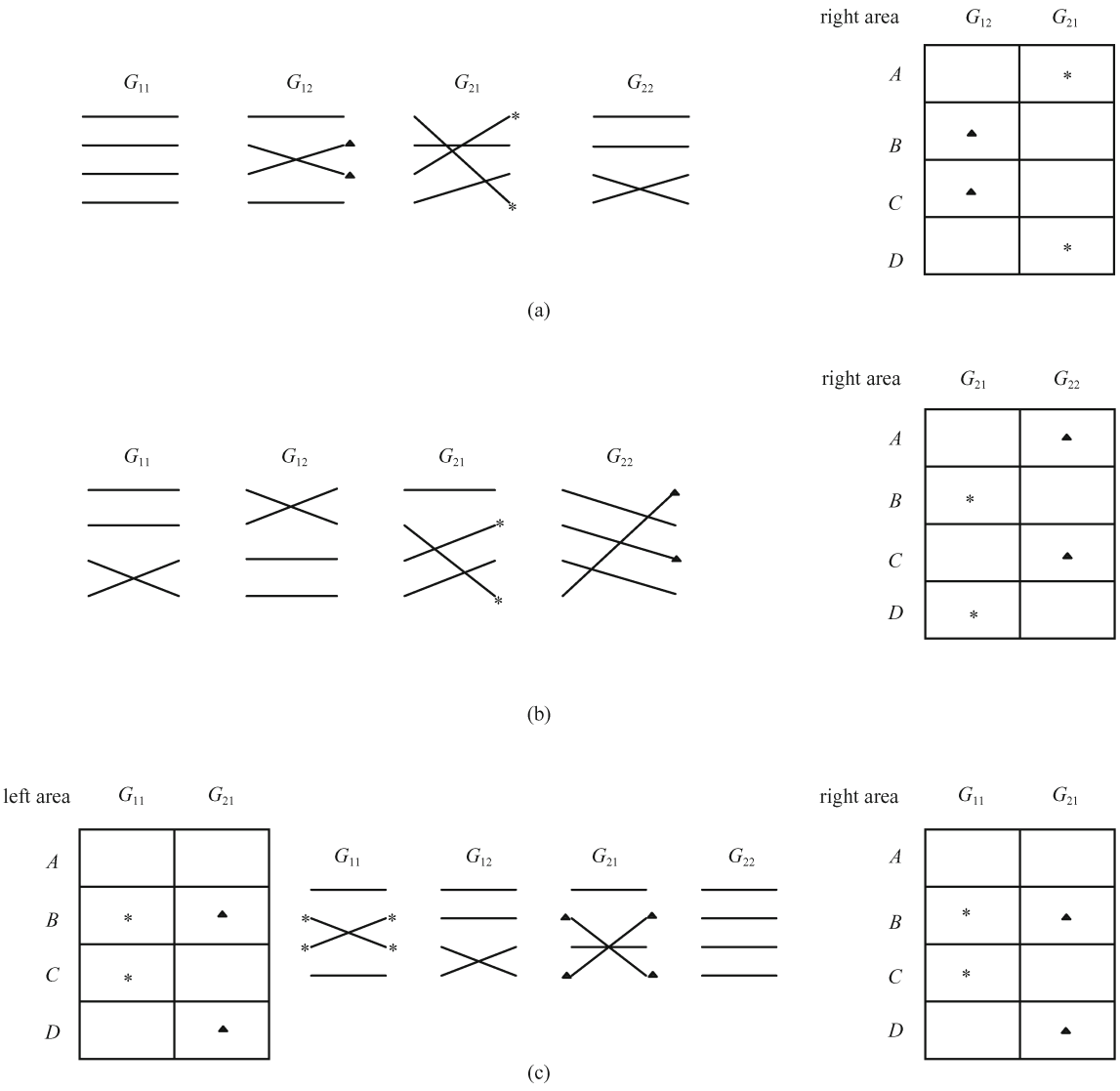


Fig. 10 Symmetry of graph G. (a) Right symmetric G; (b) another right symmetric G; (c) non-symmetric G

edges are located in pairs in right area A, D and/or B, C, as shown in Fig. 10(a).

4.2 Two basic assignment algorithms

Algorithm 1 Available for symmetric G in Cases 2 and 3 as well as in Cases 0 and 1

The algorithm idea is shown in Fig. 11. First, we assign Boolean variable X with a distinct value for sub-graph groups (G_{11},G_{12}) and (G_{21},G_{22}). Second, we assign Boolean variable Y with a distinct value for sub-graph groups (G_{11},G_{22}) and (G_{12},G_{21}). Third, Boolean variable Z with a distinct value for right area group (A,D) and (B,C) is assigned. Figure 11 gives an example.

Note When there are two pairs of cross edges in G_{ij}, the assignment is the same as G_{ij} without a crossing edge. From Fig. 11, it is easy to see that the result of assignment

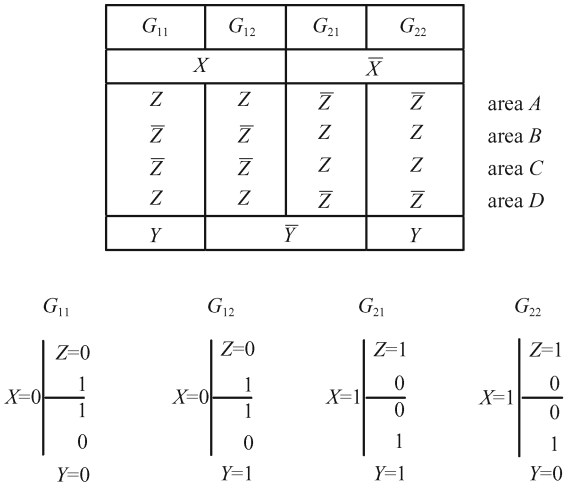


Fig. 11 Assignment Algorithm 1 and example

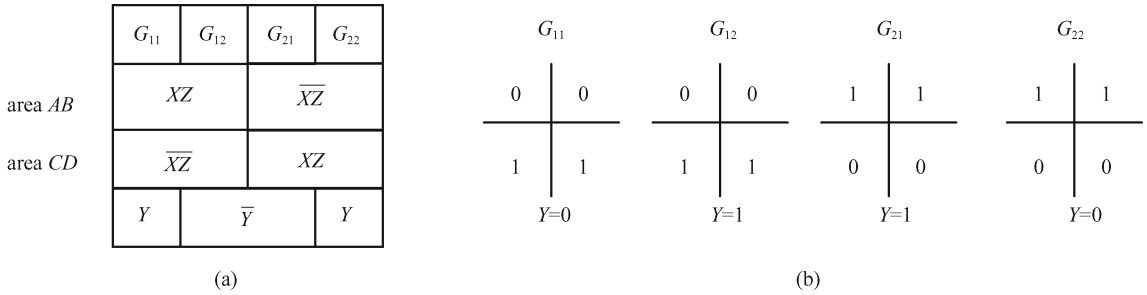


Fig. 12 Assignment Algorithm 2 and example

XYZ value for 16 edges is not unique. Regardless of the assignment realized, the XYZ values for each pair of cross edges are always the same.

Theorem 1 In Cases 0 and 1, using Algorithm 1 to graph G, the assignment values will agree with a balanced tree model.

Proof The proving process is not difficult.

Algorithm 2 Only available for G in Case 4

In G_{11}, G_{12}, G_{21} and G_{22} the Boolean variables X, Y, Z are assigned, as illustrated in Fig. 12. Figure 12(a) shows a general assignment, and Fig. 12(b) shows an example of an assignment. The Boolean variable X and Z values depend on the left or right end position of each edge.

Theorem 2 In Case 4, using Algorithm 2 to sub-graph G_{ij}, the assignment values will agree with the model of a balanced tree.

Proof The proving process is not difficult.

Note In Case 4, there are different values for one pair of crossing edges in G_{ij}. In groups (G_{11}, G_{22}) and (G_{12}, G_{21}), the crossing pairs have been assigned with the same value.

5 Assignment algorithms based on identical transforms

5.1 Requirement for identical transform

The identical transform means that after changing topological structure, graph G has the same answer space. For assigning non-symmetric graphs in Cases 2 and 3, we need to extend the equivalent transform principles listed in Lemmas 2 to 4.

Lemma 6 Swap two edge ends in 2-dimensional space (the identical transform 1)

Swap the right ends of two edges that locate in the same 2-dimensional (area) but not in the same 1-dimensional space and form a new graph G'. If G' has been assigned properly and enables two swapping edges with the same bit value Z, then the graph G can also be given the valid assignment.

Based on Lemma 6, the reformed graph G will make a 'crabstick' as shown in Fig. 13. Here, when the two edges e_1 and e_3 connect on the left, their right ends locate in different 1-dimensional spaces, while they locate in the same 2-dimensional space as shown in Fig. 13(a). After fixing their left ends and swapping their right ends, we can bipartite a cycle and form a new 'crabstick' composed of (e'_1, e'_3) and its length $L = 2$, as shown in Fig. 13(b). When the figure has been assigned properly, a pair of bit string XYZ values can be set to edges (e'_1, e'_3) in 'crabstick' in a convertible way, and their opposite bit Z guarantees that the edges e'_1 and e'_2 can been assigned with the same bit Z. By Lemma 6 the transform to 'crabstick' does not affect proper assignment for graph G.

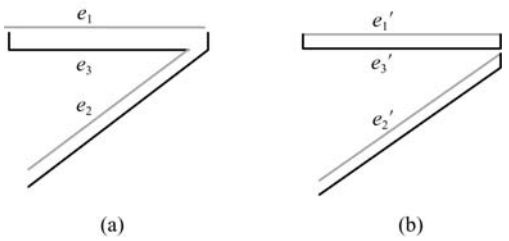


Fig. 13 Swap right two edge ends and form a 'crabstick'. (a) Part of G before swapping; (b) part of G′ after swapping

Lemma 7 Swap two edge ends in 3-dimensional space (the identical transform 2)

Fix the left ends of two disjoint edges and swap their right ends located in the same 3-dimensional space while in different 2-dimentional spaces, i.e., interchange between area A and B (or C and D) in making a new graph G'. If G' is assigned properly and two swapping edges have the same bit value YZ, then the graph G also has good assignment.

As illustrated in Fig. 14, after swapping the right ends of two edges e_1 and e_2, the new swapped edges are denoted by e'_1 and e'_2. If they are assigned with a pair of binary values XYZ and $\overline{X}YZ$ respectively, and it can be assured that graph G' is right balanced in 1- and 2-dimensional spaces, then we can respectively assign e_1 and e_2 with a pair of binary values XYZ and $\overline{X}YZ$ and retain graph G in a right balance in 1- and 2-dimensional spaces.

Lemma 8 Swap two edge ends between different 3-dimensional spaces (the identical transform 3)

Fix the left ends of two disjoint edges, swap their right ends between two quadrants (for example, from quadrant I to IV or reverse), and make a new graph G'. If G' is

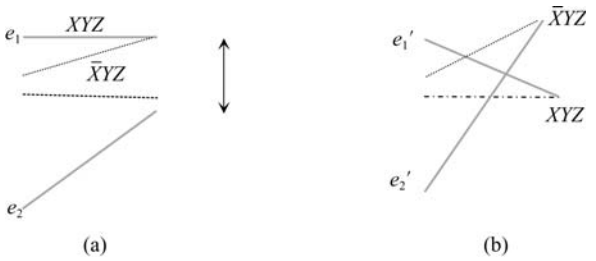


Fig. 14 Swap the end point of two edges. (a) Before swapping; (b) after swapping

assigned properly and the two swapped edges are assigned with the same bit value XYZ, then the graph G can also be assigned properly, i.e., graphs G and G′ have identical answers.

Lemmas 6–8 can also be applied to that case where the right ends of two edges are fixed and their left ends are swapped.

Unless otherwise specifically stated, in the following algorithms we suppose that the swapped cross edge is labeled by e_1 and a non-cross edge is labeled by e_2. If e_1 and e_2 locate in the same cycle, we apply Algorithm 3 or 5 to edge (e_1,e_2) to realize the identical transform. Otherwise, we execute Algorithm 4 to conduct an identical transform on the two pairs of edges.

5.2 Connection graph and identical transform

Definition 3 Distance between edges in a cycle

The distance between two edges is the hop along a cycle from one edge to another, noted by d. When d is even, we call the two edges on a cycle with an even distance, and vice-versa. If all cycle edges in graph G are two-colored or labeled alternatively by symbol '+' and '−', then the two edges on a cycle with the same color have an even distance, while those with different colors have an odd distance.

Considering that in a multi-cycle connection graph it is difficult to distinguish these interleaved cycles, we propose a loop graph representation. It is isomorphic with and equivalent to the connection graph in a topological sense. In the loop graph, the four coordinate points on the abscissa represent right areas A, B, C, and D in the connection graph, and the four coordinate points on the ordinate represent left areas A, B, C, and D. For a specific cycle, an 'edge' in the connection graph is substituted by a 'vertex' in the loop graph, where abscissa/ordinate values indicate edge end positions in the right/left area. A 'vertex' in the connection graph is substituted by a 'right-angle edge' in the loop graph, where a horizontal/vertical line (edge) indicates the left/right vertex of these edges in the connection graph. A 'crabstick' in the connection graph is substituted by two superposed vertices in the loop graph, which is labeled by a symbol '**:**'. In the loop graph each row or each column has only four vertices.

Figure 15(a) shows a loop graph that is topologically equivalent to the connection graph in Fig. 6. In this loop graph there are two cycles measuring 4 and 8 in length respectively, and another cycle which is 4 in length. However, the latter is concentrated into a vertex in the bottom-right graph, labeled by a very small rectangle. Taking alternate vertices on a cycle, we can split the loop graph into two parts, $G_1(8)$ and $G_2(8)$. As same as dividing the connection graph, we can separate the loop graph in two ways, as shown in Figs. 15(b) and 15(c). They corresponded to two sub-graphs G_1 and G_2 in Fig. 7 in the upper and lower half.

We can decompose the loop graph into four-parts by using Descartes coordinate axes. It is obvious that the cross edges in the connection graph correspond to the

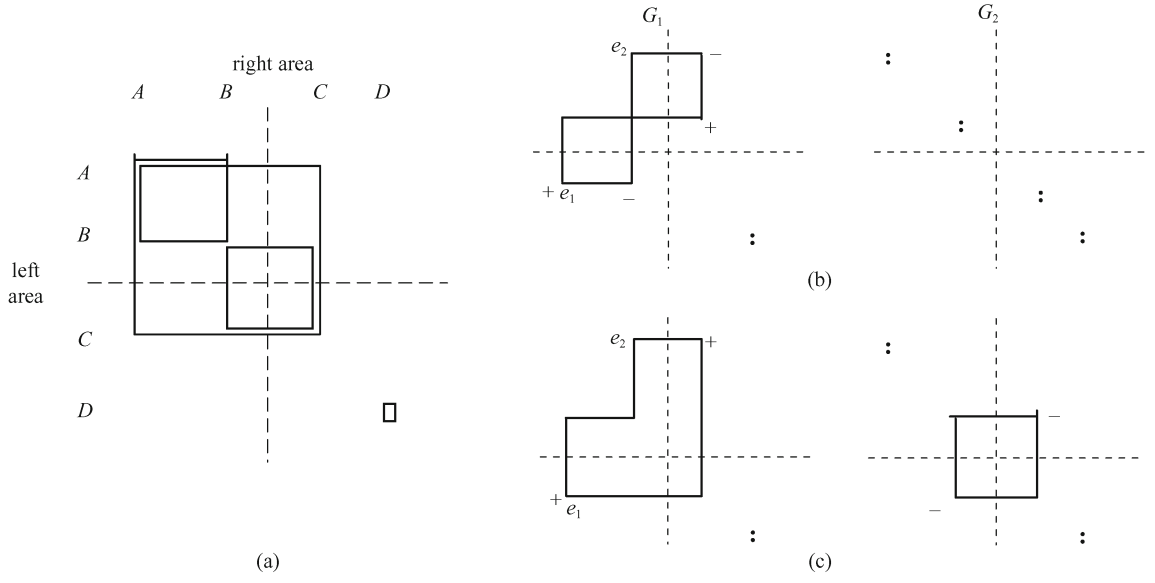


Fig. 15 Loop graph equivalent to connection graph

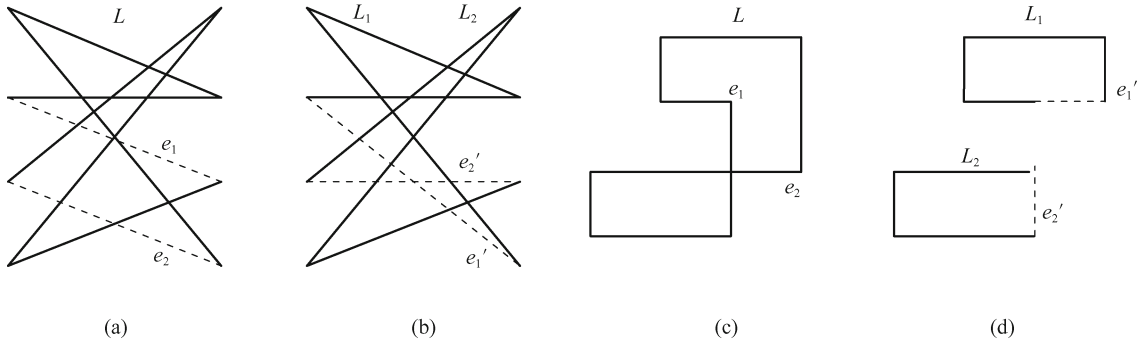


Fig. 16 To split a cycle into two parts using identical transform

vertex on quadrant I and III in the loop graph, called 'cross vertex'. The non-cross edges in the connection graph correspond to the vertex in the loop graph on quadrant II and IV, called 'non-cross vertex'.

Corollary 5 After swapping the right/left ends of two edges which have even-distance ($L \geqslant 4$) in a cycle and locate in different areas, the cycle will split into two new cycles, each containing a swapping edge.

Figures 16(a) and 16(b) show a cycle change in the connection graph after swapping. First, select two edges (e_1, e_2) with the same color and swap their right ends. The cycle L is decomposed into two small cycles L_1 and L_2, where $e'_1 \in L_1$, $e'_2 \in L_2$, while colors of the other edges remain unchanged. Because the swapped edges (e'_1, e'_2) locate in different cycles, they could be assigned with different colors. Figures 16(c) and 16(d) show the correspond change in the loop graph.

Corollary 6 After swapping the right/left ends of two edges which have odd-distances ($L \geqslant 6$) in a cycle and locate in different areas, these edges continue to locate in the original cycle with odd-distances and distinct colors. Between two edges there are two paths, in one of which each edge will change its color.

Figure 17 shows the cycle change before and after swapping in the loop graph. Here vertex e'_1 and e'_2 continue to locate in a cycle with an odd-distance and original color, although all vertices on a path at the bottom-right path between vertex e'_1 and e'_2 have changed their colors.

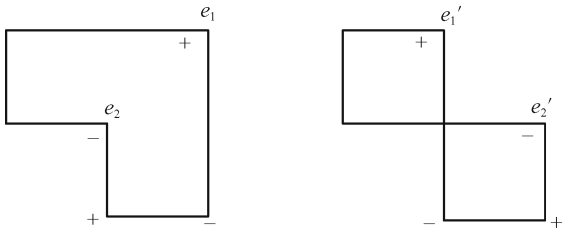


Fig. 17 Two vertices located on a cycle with odd-distance swapping across areas

Corollary 7 After swapping the ends of two edges located in different cycles, the two cycles are united, and the two edges have an even-distance (with the same color) in the new cycle. This is a converse of Corollary 5.

In the remaining sections of this paper, swapping across areas means that in the connection graph, two edges in different areas within the same quadrant swap their ends. For the loop graph, it means two vertices on the same side or the same half of the graph swap their locations.

5.3 Identical transform in a cycle with different areas

Algorithm 3 The identical transform between a cross edge and non-cross edge in the same cycle

Suppose that in a non-symmetric graph, e_1 is a cross edge that needs to alter its right end position, e_2 is a non-cross edge whose right end is located in the same 3-dimensional space but in different 2-dimensional spaces, and e_1 and e_2 belong to the same cycle in G ($L \geqslant 4$). Swapping the right ends between e_1 and e_2 can form a new right symmetric graph G', which has the same answer as G.

Step 1 Fix the left ends of edge (e_1, e_2), swap their right end positions, and form a new edge $(e'_1, e'_2) \in G$. If (e_1, e_2) locates in the same cycle in G with even-distance, according to Lemma 5, the two new cycles will be built in G', each containing e'_1 and e'_2. If (e_1, e_2) locates in the same cycle in G with odd-distance, according to Lemma 6, e'_1 and e'_2 will remain in the same cycle with odd-distance in G'.

Step 2 Take alternate edges on a cycle and split G' into G'_1 and G'_2. If the edge (e'_1, e'_2) in G' locates in the same cycle with odd-distance, they will be automatically separated into G'_1 and G'_2. If e'_1 and e'_2 locate on different cycles in G', we can consciously divide them into G'_1 and G'_2.

Step 3 Take alternate edges (or vertices) on a cycle in G'_1 and G'_2 and form the right symmetric G'_{11}, G'_{12}, G'_{21}, and G'_{22} respectively. Without loss of generality, suppose $e'_1 \in G'_{11}$, $e'_2 \in G'_{22}$.

Step 4 Apply Algorithm 1 to G', and let G'_{11} and G'_{22} be assigned with the same Y.

Step 5 Assign all unchanged edges in G with the bit values equal to G'. In addition, let $(e'_1) \Rightarrow e_1$, $(e'_2) \Rightarrow e_2$.

The non-symmetric graph shown in Fig. 15 can be solved using identical transform, as shown in Fig. 18. Here, the cross vertex e_1 resides in area A and needs to

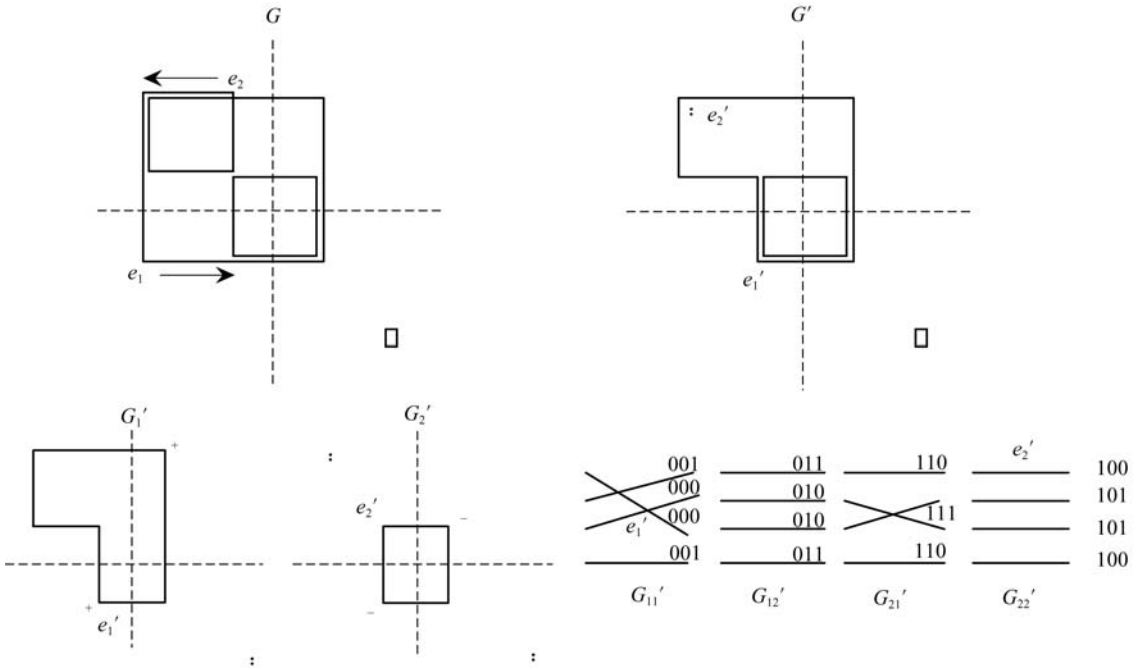


Fig. 18 Swap the right end between a cross vertex and a non-cross vertex

swap to area B, while the non-cross vertex e_2 resides in area B and needs to swap to area A. For the transformed graph G' there are two splitting methods. Let e_1' stay in G_1' and e_2' stay in G_2' to guarantee that e_1' and e_2' are assigned with the same YZ value. A set of assignments for four four-tuples is shown in the bottom-right part of Fig. 18. In this example, the XYZ value assigned for e_1' is 000, and the XYZ value assigned for e_2' is 100. Thus, they have the same YZ value.

Theorem 3 In Cases 2 and 3, if the cross vertex e_1 and non-cross vertex e_2 locate on the same cycle ($L \geqslant 4$) in different areas, using Algorithm 3 to non-symmetric graph G will certainly have a proper assignment.

Proof is omitted.

By swapping cross vertex positions (areas), Algorithm 3 realizes a graph transform from non-symmetric to symmetric, and the precondition that cross vertex e_1' and non-cross vertex e_2' must locate on the same cycle guarantees that they have been assigned with the same YZ value, and vertices e_1 and e_2 have the same YZ value. In addition, since G' has a proper assignment, G also has a proper assignment.

Figure 19 shows the final XYZ value for an example in Fig. 5. It also presents the topology of the 7-stage Ω network as well as the states of 8 switches on each stage.

5.4 Distinguishing rule about same cycle and different areas

To swap the end of cross vertex e_1 to another area, the first thing to do is ensuring that there is a non-cross vertex e_2 located on the same cycle and in another area with e_1. Hence, in non-symmetric graph G we need to distinguish if this situation could occur, and it is the precondition for using Algorithm 3. Since the cross vertex and non-cross vertex always present themselves in pairs on a cycle, we have two lemmas as follows:

Lemma 9 If there are two cross vertices in the same quadrant and on a cycle ($L \geqslant 4$), then finding a pair of cross and non-cross vertices with even distance in different areas on the cycle is certain.

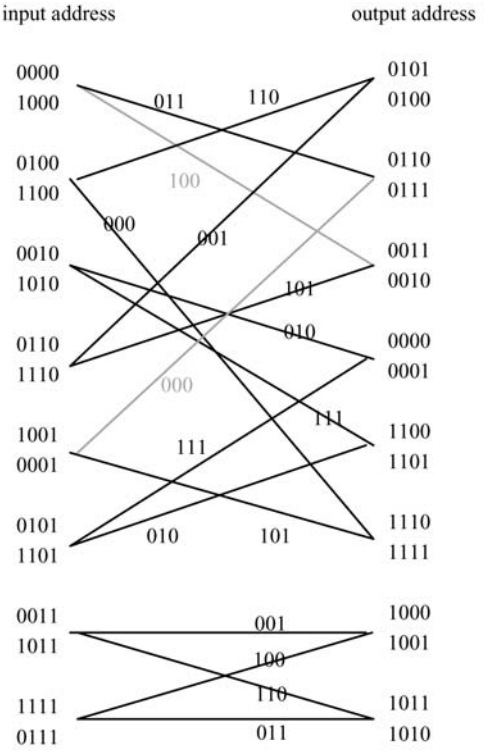
Figure 20 shows two types of loop graphs, where there are two cross vertices in one quadrant and two non-cross vertices in another quadrant. This can satisfy the requirement of Algorithm 3. Based on Lemma 9, we can deduce the following corollary.

Corollary 8 In a loop graph, when two cross vertices on a cycle locate in the same quadrant, there is no non-cross vertex e_2 which locates on the same cycle with even distance in different areas corresponding to a cross vertex e_1 only when $L = 2$ (i.e., 'crabstick').

Lemma 10 If there are two or more cross vertices which locate in different quadrants on a cycle ($L \geqslant 6$), then finding a pair of cross and non-cross vertices with even (or odd) distance in different areas on the cycle is certain.

Figures 21(a) and 21(b) show a pair of cross and non-cross vertices in different areas on a cycle ($L = 6$). Figures 21(c) and 21(d) show four cross vertices and four non-cross vertices in different areas on a cycle ($L = 8$). Because of the odd distance between any cross vertex and any non-cross vertex, so a pair of cross and non-cross vertices with odd distance in different areas on the cycle can definitely be found, satisfying the requirement of Algorithm 3. Based on Lemma 10, we can deduce the following corollary.

Corollary 9 In a loop graph when two or more cross vertices on a cycle locate in different quadrants, there is no



$$T=\begin{array}{c}\begin{array}{cccccccccccc} S_1 & S_2 & S_3 & S_4 & X & Y & Z & D_1 & D_2 & D_3 & D_4 \end{array}\\ \left(\begin{array}{cccccccccccc} 0&0&0&0&0&1&1&0&1&1&1\\ 1&0&0&1&0&0&1&0&0&1&1\\ 0&1&0&0&1&1&0&0&1&0&1\\ 1&1&0&0&0&0&0&1&1&1&1\\ 0&0&1&0&0&1&0&0&0&0&1\\ 1&0&1&0&1&1&1&1&1&0&0\\ 0&1&1&0&0&0&1&0&1&0&0\\ 1&1&1&0&1&0&1&0&0&1&0\\ 0&0&0&1&1&0&1&1&1&1&0\\ 1&0&0&1&0&0&0&0&1&1&0\\ 0&1&0&1&1&1&1&0&0&0&0\\ 1&1&0&1&0&1&0&1&1&0&1\\ 0&0&1&1&0&0&1&1&0&0&0\\ 1&0&1&1&1&1&0&1&0&1&1\\ 0&1&1&1&0&1&1&1&0&1&0\\ 1&1&1&1&1&0&0&1&0&0&1 \end{array}\right) \end{array}$$

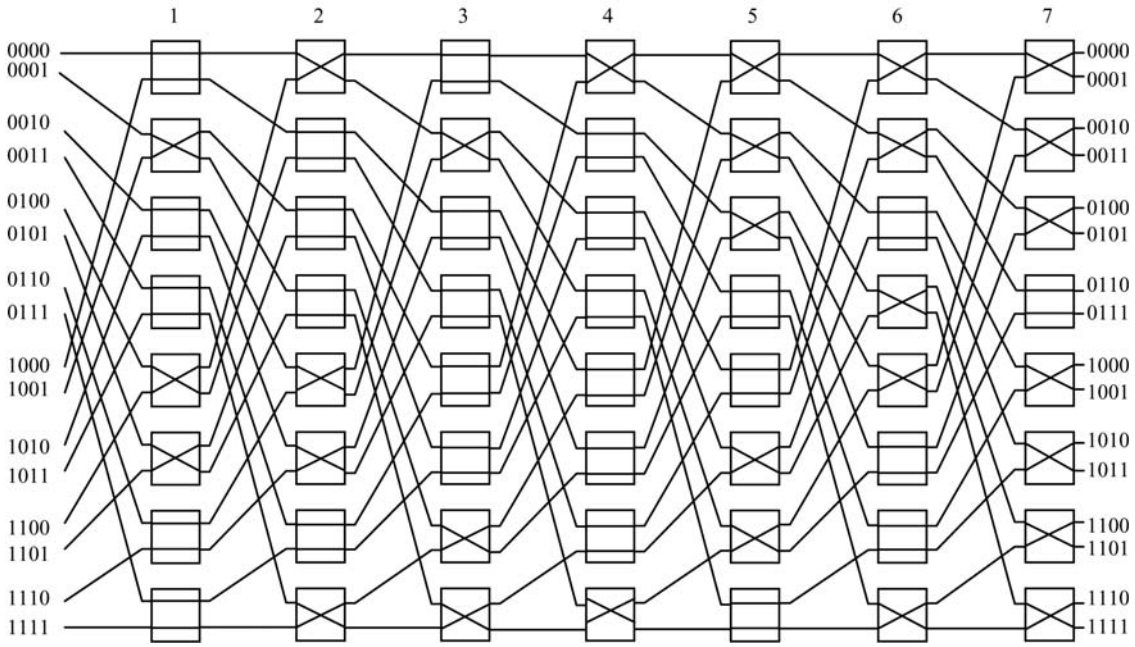


Fig. 19 Assignment for graph determines the routing on Ω network

non-cross vertex e_2 which locates on the cycle in different areas corresponding to a cross vertex e_1 only when $L = 4$.

5.5 Identical transform in different cycles

To distinguish whether the graph G is non-symmetric is the precondition to apply Algorithm 3. In Case 2, when two pairs of cross vertices on the left and/or right side concentrate in areas B and C (type 0-2-2-0) or evenly distribute in four areas (type 1-1-1-1), G is symmetric. Only when the cross vertex distribution is present in type 0-2-1-1 is G asymmetric. Thus, we can deduce the following Lemma.

Lemma 11 In Case 2, when the cross vertices distribute as Fig. 22 shows, G is asymmetric.

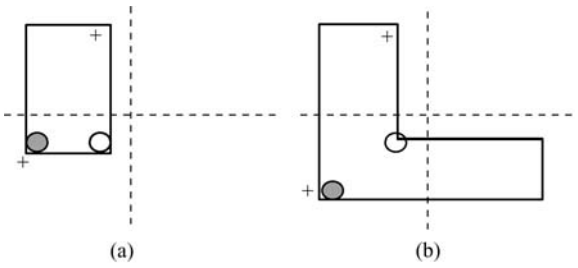


Fig. 20 Cross vertices in quadrant on the same cycle

In Fig. 22, the cross vertices distribute in the form of type 0-2-1-1 either from top to bottom or from left to right. Swapping one cross vertex with a non-cross vertex across areas can make the graph G symmetric.

In Fig. 22, for any selected cross vertex e_1, if there is no corresponding non-cross vertex on the same cycle in different areas, based on Corollary 9, there is only one possibility that a pair of cross vertices in different quadrants locates on an independent cycle with $L = 4$, as shown in Fig. 23.

For special non-symmetric graph G such as Fig. 23, we need a new algorithm. To make G symmetric, the cross vertex needs to change its area. According to Corollary 9, there is no corresponding non-cross vertex e_2 on the cycle in different areas. It belongs to another cycle. At this time Corollaries 7 and 5 are needed. First, we need swap the ends of vertices e_1 and e_2 which locate in different cycles. Second, swap the ends of a pair of non-cross vertices on a new cycle with even distance, so that the e_1 and e_2 still locate on different cycles. In summary, we need to swap two pairs of vertices which locate on two cycles simultaneously.

Algorithm 4 Identical transform for two pairs of edges on different cycles

Step 1 In Fig. 24, suppose that the cross edge e_1 needs to swap the right end and it joint with non-cross edge e_3 in the right area B. Select a pair of non-cross edges e_2 and e_4 which joint in the right area A. These four edges locate on two different cycles in G.

Step 2 Swap the right ends of edges e_2 and e_4 into area B, keep them jointed and label the swapped edges as e'_2 and e'_4. Keep edges e_1 and e_3 jointed, swap them into area A, and label the swapped edges as e'_1 and e'_3.

Step 3 Take alternate edges on a cycle in G' and divide it into two parts. Because G' is composed of two or more cycles, there are two divided methods at least. Without loss of generality, we let $(e'_1, e'_2) \in G'_1$, $(e'_3, e'_4) \in G'_2$.

Step 4 Split G'_1 and G'_2 and form right symmetric G'_{ij}. In G'_{ij} the edges e'_1 and e'_2 have two possible locations.

1) If e'_1 and e'_2 locate on different cycles in G'_1, then their locations in G'_{ij} depend on e'_3 and e'_4. If e'_3 and e'_4 locate in G'_{22}, let e'_1 and e'_2 locate in G'_{11}. If e'_3 locates in G'_{21}, and e'_4 in G'_{22}, let e'_1 and e'_2 locate in different G'_{1j}.

2) If e'_1 and e'_2 locate on the same cycle with odd (even) distance in G'_1, then they locate on the same (different) G'_{1j}, and e'_3 and e'_4 also locate on the same (different) G'_{2j}.

Step 5 Call Algorithm 1 to assign G'. Then the value Y of edge e'_i $(i = 1,2,3,4)$ are all the same or pairwise the same, which means that e'_1 and e'_4 have the value Y, and e'_3 and e'_2 have the same or different value Y.

Step 6 Assign G. Let $(e'_i) \Rightarrow e_i$ $(i = 1,2,3,4)$. The unchanged edges in G have the same value as those in G'.

Figure 25 is an example of using Algorithm 4. It shows the change of graph G after swapping. Here, four swapped edges centralize in two four-tuples, and they are assigned with the same Y.

Theorem 4 In the non-symmetric graph in Case 2, applying Algorithm 4 to two cycles and taking the identical transform, (e'_1, e'_4) and (e'_2, e'_3) are definitely assigned with the same YZ value, and G has good assignment.

Proof When graph G is composed of two or more cycles, as shown in Fig. 23, it is possible that four cross vertices concentrate in G_1. The loop graph G_1 contains 8 vertices. Thus, two cross vertices certainly locate in different rows and/or columns on the same cycle with two non-cross vertices which locate in another area. As shown in Fig. 26, the dashdotted line is the cycle in G; real line is the cycle in G_1. It is very important for applying Algorithm 4 to deal with asymmetric graphs in Case 2 that to let e_1 and e_2 reside in the same cycle in G_1.

Because vertices e_1 and e_2 reside on a cycle in G_1 with even distance, according to Corollary 5, after swapping them across areas, the corresponding vertices e'_1 and e'_2 locate on different cycles in G'_1. At this time they can be placed either all in G'_{11}, or e'_1 in G'_{11} and e'_2 in G'_{12}, depending on the location of e'_3 and e'_4 in G'_2. To assign them with

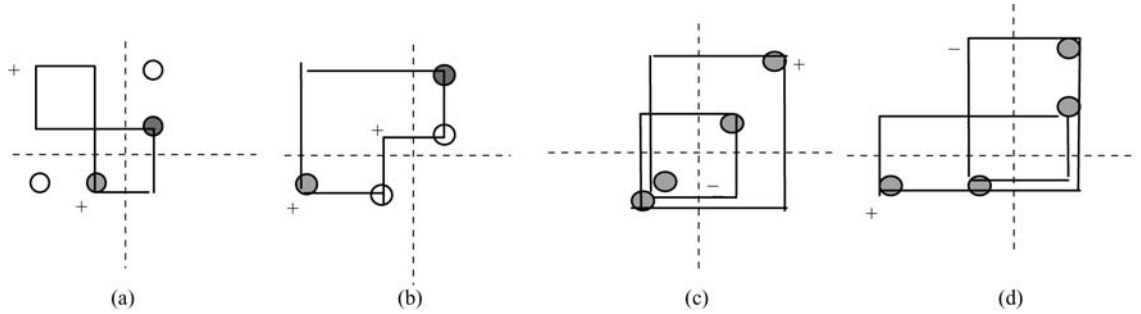


Fig. 21 Cross vertices in different quadrants on the same cycle

Fig. 22 Cross vertex distribution in asymmetric loop graph ($K = 2$)

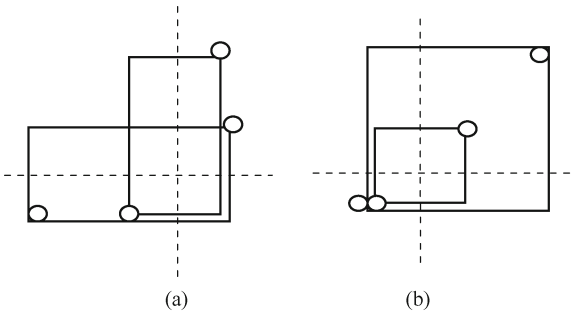


Fig. 23 Non-symmetric graph needing special process (in Case 2)

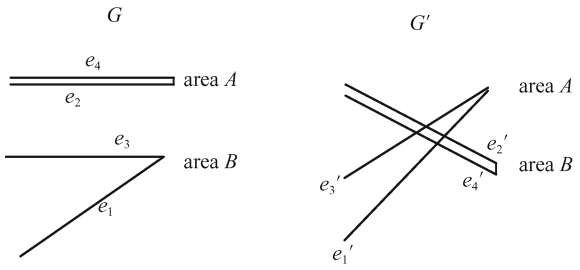


Fig. 24 Identical transform for two pair edges

the same Z, e'_1 and e'_4 should be in different four-tuples in different areas. Similarly, e'_2 and e'_3 should be in different four-tuples in different areas. Because e'_1 and e'_2 never locate on a cycle in G'_1, the possibility in Algorithm 4 (2) can be avoided.

Now let's prove that assignment Y for four edges is the same in a pair if $(e'_1, e'_2) \in G'_{11}$, $(e'_3, e'_4) \in G'_{22}$, let these two four-tuples have the same Y (see Fig. 27). Otherwise, let two four-tuples where e'_1 and e'_4 are located have the same Y, and let two four-tuples in which e'_2 and e'_3 are located have other Y.

5.6 Identical transform in a cycle with different quadrants

Before discussing asymmetric graphs in Case 3, the location of three pairs of cross vertex in a loop graph is reviewed. They reside in two areas in a quadrant by type 0-3 or 1-2. In the view of loop graph G from top-bottom or left-right, their distribution can be type 0-3-3-0, 1-2-2-1, or 0-3-2-1. A graph with type 0-3-3-0 is certainly symmetric, and one with type 0-3-2-1 is certainly asymmetric, while one with type 1-2-2-1 is possibly symmetric or asymmetric. A symmetric graph means that it is neither left symmetric nor right symmetric. Therefore, in view of G from any side, it cannot be type 0-3-3-0. Only when it demonstrates a 1-2-2-1 or 0-3-2-1 type from a top-bottom or left-right view of G can it be concluded that G may be asymmetric with six compositions.

Lemma 12 In case 3, G is asymmetric in 6 types as shown in Fig. 28.

When two-coloring G in Case 3, three pairs of cross vertices cannot reside in G_1, otherwise G would be classified into Case 1. Therefore, there must be a pair of cross vertices labeled '1' in Fig. 28 and colored differently from

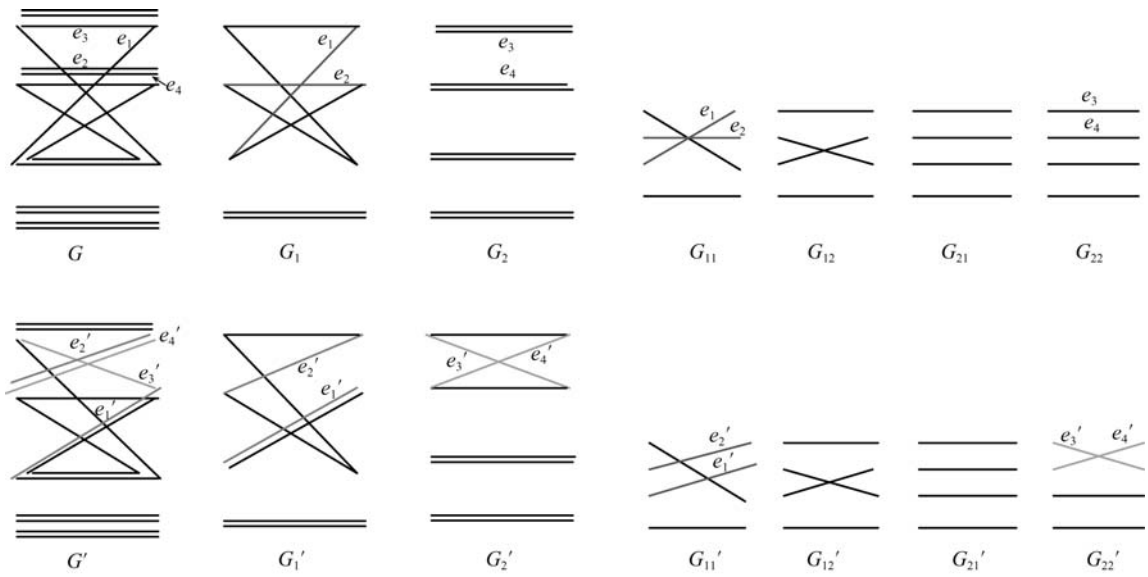


Fig. 25 Swap ends of two pairs of edges with the same Y and the same Z two by two

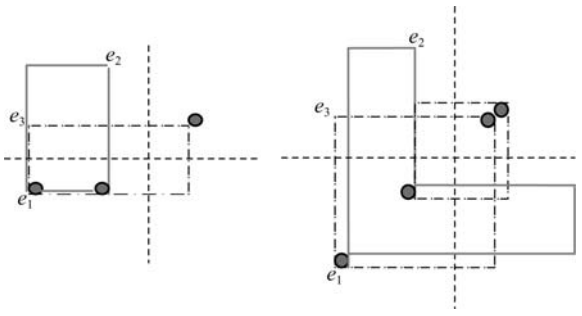


Fig. 26 Identical transform for two pairs of vertices

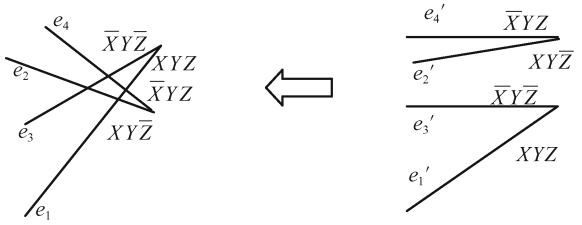


Fig. 27 Assignment Y for four edges

other two pairs. In viewing loop graph G from top-bottom or left-right in Fig. 28(a), cross vertices are present as type 0-3-2-1, and thus G is asymmetric. In Figs. 28(b) and 28(c), cross vertices are present as mixed type, i.e., to view G from a top-bottom perspective, it is present as type 0-3-2-1, while from left-right perspective, it is present as type 1-2-2-1. When a pair of cross vertices labeled '1' is present as Figs. 28(b) and 28(c), the remaining two pairs are then similar to Fig. 22(b). Thus, G is asymmetric even when viewed from a left-right perspective. In Figs. 28(d), 28(e) and 28(f), cross vertices are present as type 1-2-2-1. When a pair of cross vertices labeled '1' is present as the above figures show, the remaining two pairs are similar to Figs. 22(a), 22(b) and 22(b) respectively. Therefore, G is asymmetric whether viewed from a top-bottom or left-right perspective.

In general, using Algorithm 3 can deal with asymmetric G in Case 3. But for a given cross vertex e_1, if there is no non-cross vertex e_2 which locates in the same cycle in different areas, based on Lemmas 8 and 9, there are only two possibilities. One is that the two cross vertices locate in the same position or on a cycle $L = 2$. The other is that the two cross vertices locate in different quadrants on a cycle $L = 4$. Because cross vertices on a cycle are always present in pairs, two cross vertices located in the same quadrant on a cycle $L = 2$ means that the third cross vertex can only compose a cycle $L = 4$ with a cross vertices in other quadrants. Hence, in asymmetric G there is at least a cycle $L = 4$, where two cross vertices and two non-cross vertices locate in each quadrant. At this time, we can use Lemma 8 (the identical transform 3) for assignment as follows.

Algorithm 5 Identical transform for two pairs of non-cross edges on the same cycle

Step 1 Suppose that an asymmetric graph G in Case 3 presents as Fig. 28 shows. Select a pair of non-cross edges e_1 and e_2 on the same cycle with even distance in the connection graph, where e_1 traverses quadrant II and I and e_2 traverses quadrant III and IV. Fix their left ends and swap their right ends, making a pair of new cross edges e_1' and e_2', which locates on different cycles in G'.

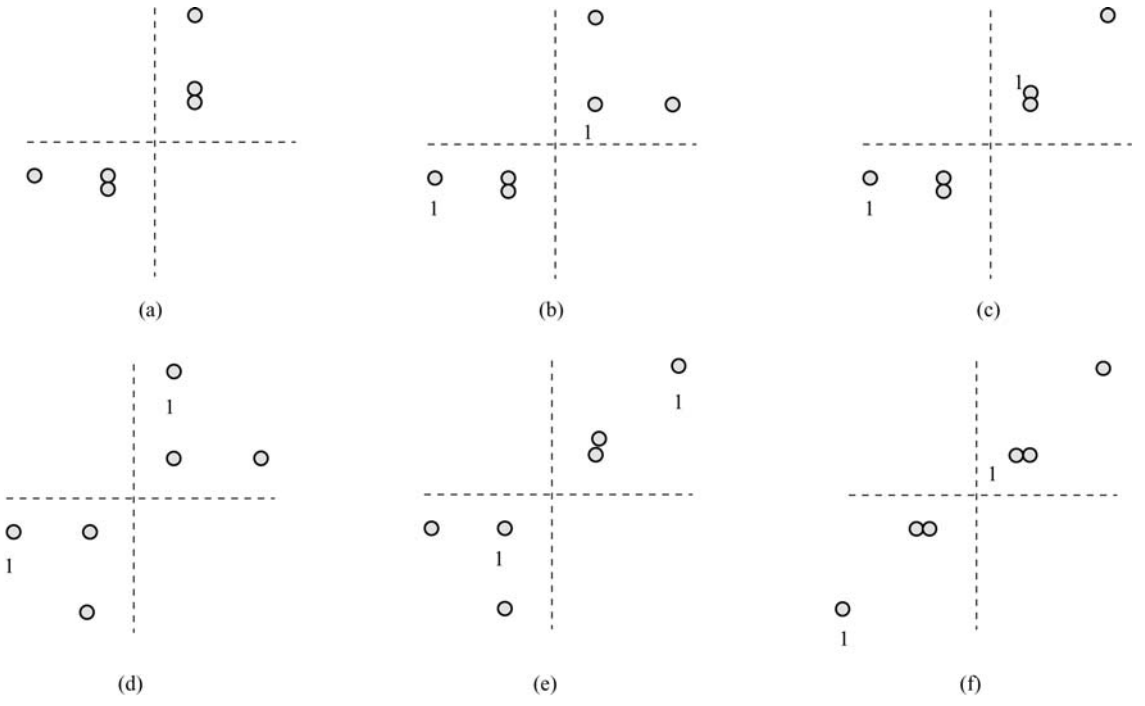


Fig. 28 Cross vertices in loop graph of asymmetric G in Case 3

Step 2 Take alternate edges on the cycle and divide G' into two parts. Let e'_1 be in G'_1 and e'_2 in G'_2.

Step 3 Continue to take alternate edges on the cycle and divide G'_1 and G'_2 into two parts, which makes four G'_{ij}, each with a pair of cross edges. Without loss of generality, suppose $e'_1 \in G'_{11}$, $e'_2 \in G'_{22}$.

Step 4 Call Algorithm 2 for assignment G' and let G'_{11} and G'_{22} have the same Y, thus guaranteeing that edge e'_1 and e'_2 have the same XYZ.

Step 5 Let $(e'_1) \Rightarrow e_1$, $(e'_2) \Rightarrow e_2$, then the unchanged edges in G have the same value as those in G'.

An example in Fig. 29 indicates that for asymmetric graph G, Algorithm 3 is not applicable, while only Algorithm 5 can be used for assignment.

Theorem 5 In Case 3, applying Algorithm 5 has a reasonable assignment for asymmetric graph G.

Proof omitted.

After using Algorithm 5 and swapping, G' in Case 3 does not fade into Case 2. First, there are no three cycles of which $L = 4$, with a pair of cross vertices on different quadrants. Otherwise, in bipartition of G three pairs of cross vertices can be concentrated in G_1, which further bipartite into G_{11} and G_{12}. Thus, making two four-tuples become type '$\# \times$', which contradicts the precondition of Case 3. Lemma 9 explains that when there are two cross vertices in the same quadrant on a cycle where $L \geqslant 4$, Algorithm 3 is available. So only in one case where using Algorithm 5 for that G there is a cycle $L = 4$ and contains a pair of cross vertices, and the remaining four cross vertices centralize into two points, as shown in Fig. 30. After selecting non-cross vertices e_1 and e_2 and using Algorithm 5, vertices e'_1 and e'_2 in G' with original cross vertex centralize together. When dividing G', the locations of cross vertices in G'_1 and G'_2 are identical.

6 Conclusions

In telecommunication switches, the rearrangeability of shuffle exchange networks is a key problem. Using the representation of connection graph and loop graph and taking alternate bipartition and compressing methods of graph, this paper presents a non-blocking routing for a 16 × 16, 7-stage Ω network. Although there are 2.1×10^{13} possible permutations in total, we classify them into five types and bring forward five algorithms for assignment. In

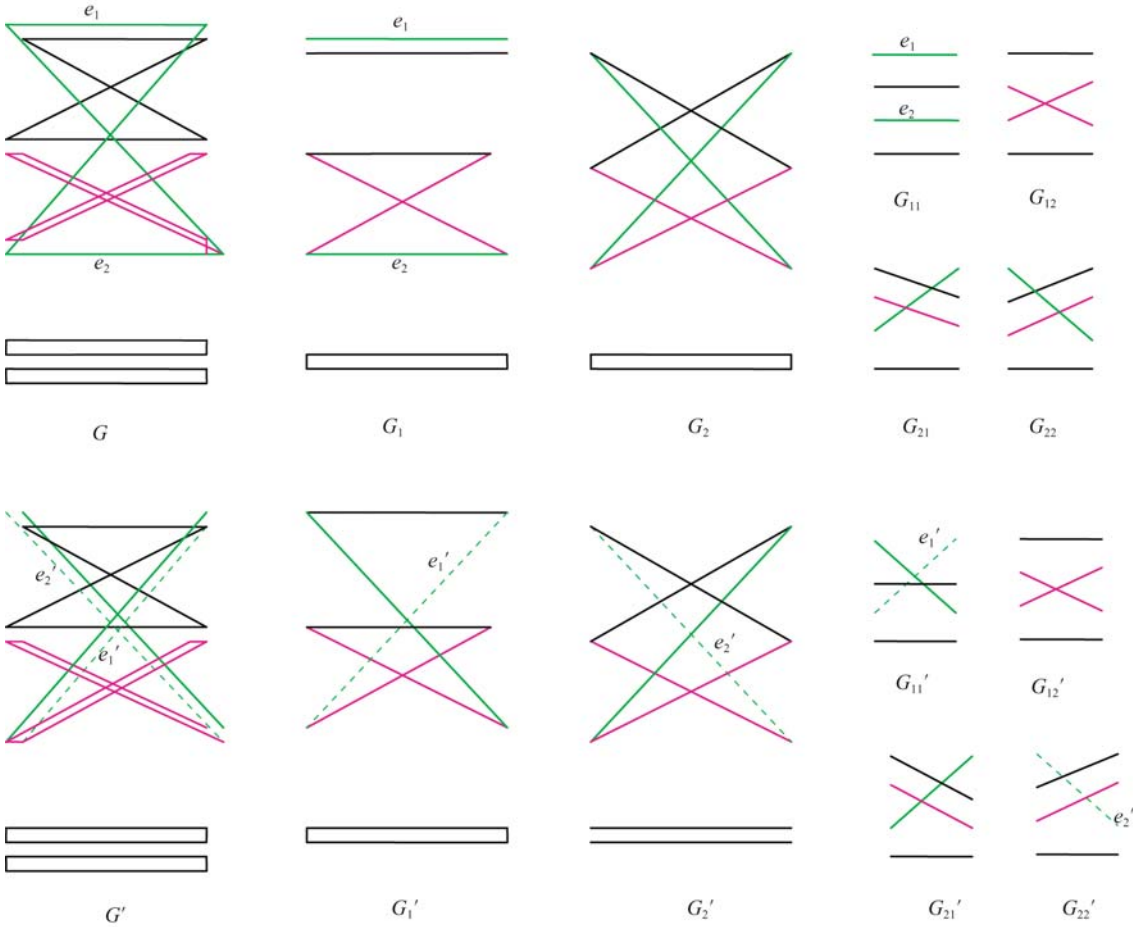


Fig. 29 Swap two non-cross edges into cross edges

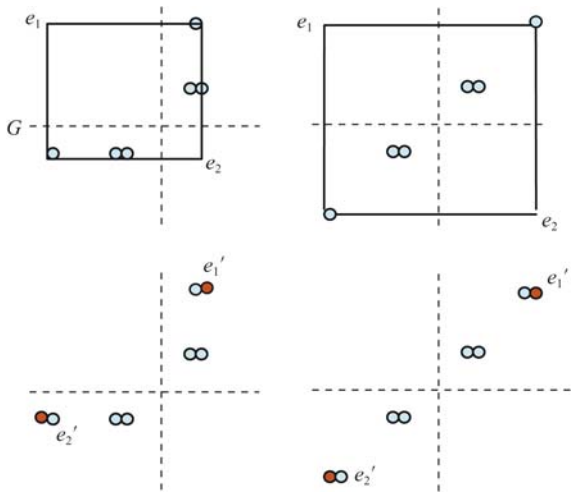


Fig. 30 Cross vertex location in G available for Algorithm 5

Cases 2 and 3, priority is taken in selecting Algorithms 1 and 3. Algorithms 4 and 5 are not available for symmetric graphs, and they act as supplements of Algorithm 3 and the last means for assignment asymmetric graph. In practice, in using identical transform, any asymmetric graph can be changed to symmetric. Virtually, this is to rearrange partnerships of cross edges and/or non-cross edges in four-tuples.

How to extend assignment algorithms to multi-stage network Ω in case of stage $n \geqslant 5$ using the classification idea and identical transform outlined in this paper is still a challenging topic. When $n \geqslant 5$, the permutation types will increase. Thus, it requires finding recursive constitution algorithms to prove the rearrangeability of a $(2n-1)$-stage shuffle exchange network.

Acknowledgements This work was supported by the National Science Foundation of USA (Grant No. 9810692), University of Missouri-Kansas City Faculty Research Grant (UMKC FRG) (No. K-2-11678).

References

1. Kim K, Raghavendra C S. A simple algorithm to route arbitrary permutations on 8-input 5-stage shuffle/exchange network. In: Proceedings of the 5th International Parallel Processing Symposium. 1991, 398–403
2. Raghavendra C S, Varma A. Rearrangeability of the 5-stage shuffle/exchange network for $N = 8$. In: Proceedings of 1986 International Conference on Parallel Processing. 1987, 119–122
3. Stone H S. Parallel processing with the perfect shuffle. IEEE Transactions on Computers, 1971, C-20(2): 153–161
4. Lawrie D H. Access and alignment of data in an array processor. IEEE Transactions on Computers, 1975, C-24(12): 1145–1155
5. Raghavendra C S. On the rearrangeability conjecture of $(2\log_2 N - 1)$-stage shuffle/exchange network. IEEE Computer Society Technical Committee on Computer Architecture Newsletter. 1994, 95, 10–12
6. Wu C L, Feng T Y. On a class of multistage interconnection networks. IEEE Transactions on Computers, 1980, C-29(8): 694–702
7. Wu C L, Feng T Y. The universality of the shuffle-exchange network. IEEE Transactions on Computers, 1981, C-30(5): 324–332
8. Benes V E. Proving the rearrangeability of connecting networks by group calculations. Bell System Technology Journal, 1975, 54: 421–434
9. Parker D S. Notes on shuffle/exchange-type switching networks. IEEE Transactions on Computers, 1980, C-29(3): 213–222
10. Linial N, Tarsi M. Efficient generation of permutations with the shuffle/exchange network. Technical Report, UCLA Computer Science Department, 1982
11. Sovis F. Uniform theory of the shuffle-exchange type permutation networks. In: Proceedings of the 10th Annual International Symposium on Computer Architecture, 1983, 185–191
12. Lee K Y. On the rearrangeability of $2(\log_2 N) - 1$ stage permutation networks. IEEE Transactions on Computers, 1985, C-34(5): 412–425
13. Andresen S. The looping algorithm extended to base 2^i rearrangeable switching networks. IEEE Transactions on Communications, 1977, COM-25(10): 1057–1063
14. Huang S T, Tripathi S K. Finite state model and compatibility theory: new analysis tools for permutation networks. IEEE Transaction on Computers, 1986, C-35(7): 591–601
15. Varma A, Raghavendra C S. Rearrangeability of multistage shuffle/exchange networks. IEEE Transactions on Communications, 1988, COM-36(10): 1138–1147
16. Linial N, Tarsi M. Interpolation between bases and the shuffle exchange network. European Journal of Combinatorics, 1989, 10(1): 29–39
17. Feng T Y, Seo S W. A new routing algorithm for a class of rearrangeable networks. IEEE Transactions on Computers, 1994, 43(11): 1270–1280
18. Sovis F. On rearrangeable networks of the shuffle-exchange type. Computers and Artificial Intelligence, 1988, 7(4): 359–373
19. Can H, Fortes J A. Rearrangeability of shuffle/exchange network. In: Proceedings of Frontiers of Massively Parallel Computation. 1990, 303–314
20. Abdennadher A, Feng T Y. On rearrangeability of Omega-Omega networks. In: Proceedings of International Conference on Parallel Precessing. 1992, 159–165
21. Kim M K, Yook H, Maeng S R. On the correctness of inside-out routing algorithm. IEEE Transactions on Computers, 1997, 46(7): 820–823
22. Cam H. Rearrangeablity of $(2n-1)$-stage shuffle-exchange networks. Society of industrial and applied mathematics. Journal of Computers, 2003, 32(3): 557–585