IEEE Access
Multidisciplinary ⋮ Rapid Review ⋮ Open Access Journal

Received 7 July 2022, accepted 31 July 2022, date of publication 8 August 2022, date of current version 11 August 2022.

Digital Object Identifier 10.1109/ACCESS.2022.3197288

RESEARCH ARTICLE

Set-to-Set Disjoint Paths Problem in Möbius Cubes

HIROYUKI ICHIDA[1] AND KEIICHI KANEKO[2], (Member, IEEE)
[1]Graduate School of Engineering, Tokyo University of Agriculture and Technology, Tokyo 184-8588, Japan
[2]Institute of Engineering, Tokyo University of Agriculture and Technology, Tokyo 184-8588, Japan

Corresponding author: Keiichi Kaneko (k1kaneko@cc.tuat.ac.jp)

This work was supported in part by the Grant-in-Aid for Scientific Research (C) of the Japan Society for the Promotion of Science under Grant 19K11887 and Grant 20K11729.

ABSTRACT The set-to-set disjoint paths problem is to find n vertex-disjoint paths $\boldsymbol{s}_i \rightsquigarrow \boldsymbol{t}_{j_i}$ $(1 \leq i \leq n, \{j_1, j_2, \ldots, j_n\} = \{1, 2, \ldots, n\})$ between two sets of vertices $S = \{\boldsymbol{s}_1, \boldsymbol{s}_2, \ldots, \boldsymbol{s}_n\}$ and $T = \{\boldsymbol{t}_1, \boldsymbol{t}_2, \ldots, \boldsymbol{t}_n\}$ in a graph whose connectivity is equal to n. It is a very important problem as well as the vertex-to-vertex disjoint paths problem and the vertex-to-set disjoint paths problem. In this paper, we propose an algorithm that generates n vertex-disjoint paths between two vertex sets in an n-Möbius cube. Much attention has been attracted by a Möbius cube because its diameter is almost half of that of a hypercube while it can interconnect the same number of vertices as the hypercube. We also give a proof of correctness of the algorithm and estimate that the time complexity of the algorithm is $O(n^6)$ and the maximum length of the paths generated by the algorithm is $2n - 2$.

INDEX TERMS Fault tolerant systems, multiprocessor interconnection networks, network topology, parallel processing, supercomputers.

I. INTRODUCTION

In these few decades, parallel computers, especially massively parallel systems have been studied enthusiastically. Since many processing elements or vertices are connected in a massively parallel system, an efficient interconnection network is crucial. Hence, many topologies [1], [2], [3], [4] have been proposed for interconnection networks and intensively studied [5], [6], [7], [8], [9], [10], [11], [12], [13] to take the place of conventional topologies such as the mesh, the torus, and the hypercube [14]. A Möbius cube [15] is one of such new topologies. Much attention has been attracted by a Möbius cube because its diameter is almost half of that of a hypercube while it can interconnect the same number of vertices as the hypercube [16], [17], [18], [19], [20], [21].

In this paper, we follow the traditional definitions and notations in the graph theory. A graph G consists of a pair of a vertex set V with $|V| > 0$ and an edge set $E(\subset V \times V)$ with $|E| \geq 0$. If there is an edge $(\boldsymbol{x}, \boldsymbol{y})$ in G, the vertices $\boldsymbol{x}$ and $\boldsymbol{y}$ are adjacent and the edge $(\boldsymbol{x}, \boldsymbol{y})$ is incident to $\boldsymbol{x}$ and $\boldsymbol{y}$. For two graphs $G(V, E)$ and $G'(V', E')$, G' is called a sub graph of G if $V' \subset V$ and $E' \subset E$ hold. For a vertex sequence P: $\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_k$ with $\boldsymbol{x}_i$ and $\boldsymbol{x}_{i+1}$ $(1 \leq i \leq k-1)$ are all adjacent, P is called a path and the length of the path is $k-1$. In this paper, we denote the path by $\boldsymbol{x}_1 \rightarrow \boldsymbol{x}_2 \rightarrow \cdots \rightarrow \boldsymbol{x}_k$ or $\boldsymbol{x}_1 \rightsquigarrow \boldsymbol{x}_k$, and an edge $(\boldsymbol{x}, \boldsymbol{y})$ by $\boldsymbol{x} \rightarrow \boldsymbol{y}$. If the path P contains only one vertex, it is regarded a path of length 0. For two vertices $\boldsymbol{x}, \boldsymbol{y}(\in G)$, the smallest length of the path between them is called the distance between $\boldsymbol{x}$ and $\boldsymbol{y}$. The largest value among the distances of pairs of vertices in a graph G is called the diameter of G. A graph G is called connected if there is a path between any pair of vertices in G. A graph G is called disconnected if G is not connected. For a connected graph G, the smallest k such that deleting k vertices from G makes the residual graph disconnected is called vertex connectivity (or connectivity in short) of G. For two paths P: $\boldsymbol{x}_1 \rightarrow \boldsymbol{x}_2 \rightarrow \cdots \rightarrow \boldsymbol{x}_k$ and Q: $\boldsymbol{y}_1 \rightarrow \boldsymbol{y}_2 \rightarrow \cdots \rightarrow \boldsymbol{y}_l$, P and Q are called vertex-disjoint (or disjoint in short) if $\{\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_k\} \cap \{\boldsymbol{y}_1, \boldsymbol{y}_2, \ldots, \boldsymbol{y}_l\} = \emptyset$.

We pick up the Möbius cube, which is one of the cube-based topologies. The hypercube is the representative of the cube-based topologies. The Möbius cube has an almost

The associate editor coordinating the review of this manuscript and approving it for publication was Xujie Li.

half diameter of that of the hypercube of the same size. Also, we can apply recursive algorithms to the Möbius cube because it has the recursive structure. In addition, the Möbius cube is practical because it has a smaller average distance compared to the other cube-based topologies of the same size. Furthermore, the Möbius cube has a simple structure. Duato *et al.* [22] insist that "Simple designs of topologies often lead to higher clock frequencies and may achieve higher performance. Moreover, customers appreciate networks that are easy to understand because it is easier to exploit their performance."

The set-to-set disjoint paths problem is one of the unsolved problems in Möbius cubes, and it is defined as follows: For a source-vertex set $S = \{s_1, s_2, \ldots, s_n\}$ and a target-vertex set $T = \{t_1, t_2, \ldots, t_n\}$ in an n-connected graph $G(V, E)$, generate n paths P_i: $s_i \rightsquigarrow t_{j_i}$ $(1 \leq i \leq n)$ such that $\{j_1, j_2, \ldots, j_n\} = \{1, 2, \ldots, n\}$ and the paths P_i's are vertex-disjoint, or simply disjoint. The problem of the set-to-set disjoint paths is an important issue [23], [24], [25], [26], [27], [28], [29] as well as the problems to generate vertex-to-vertex disjoint paths [30], [31], [32], [33], [34], [35], [36], [37], [38], vertex-to-set disjoint paths [18], [39], [40], [41], [42], [43], [44], [45], [46], [47], [48], [49], and pairwise disjoint paths [50], [51], [52], [53], [54].

Finding disjoint paths between two vertex sets in a massively parallel system establishes secure communication paths that do not interfere each other. Also, the algorithm that solves the set-to-set disjoint paths problem can be applied to solve the vertex-to-vertex and vertex-to-set disjoint paths problems to attain the full-bandwidth communication. In addition, it is inevitable to operate a massively parallel system under the existence faulty vertices and/or edges. Hence, finding multiple disjoint paths between two vertices augments the probability to establish a fault-free path between them.

Regarding an n-dimensional Möbius cube or MC_n in short, Kocík and Kaneko [34] have proposed an algorithm that solves the vertex-to-vertex disjoint paths problem in $O(n^2)$ time, and the maximum length of the paths generated by the algorithm is $3n - 5$. Moreover, Kocík *et al.* [18] have proposed an algorithm that solves the vertex-to-set disjoint paths problem in $O(n^4)$ time, and the maximum length of the generated paths is $2n - 1$.

As it is widely known, from the Menger's theorem [55], the maximum-flow algorithm can solve the problem of the set-to-set disjoint paths in an arbitrary graph $G(V, E)$ in a polynomial-order time of $|V|$. Nevertheless, if we apply the maximum-flow algorithm to MC_n, its time complexity is impractically large since MC_n has 2^n vertices. In this paper, we extend the previous work by Kaneko [56] and propose an algorithm called S2S (set-to-set), which solves the problem in polynomial-order time of n instead of 2^n. The algorithm is divided into three cases depending on the distribution of the source vertices and the target vertices. For a source-vertex set S and a target-vertex set T with $|S| = |T| = n$ in MC_n, Algorithm S2S generates n disjoint paths between S and T. We also present the results including the average performance by a computer experiment.

Recently, Kaneko *et al.* [57] proposed a generic algorithm that solves the set-to-set disjoint paths problem in many cube-based topologies, which include the Möbius cube. The algorithm takes $O(n^7)$ time in MC_n, and the maximum length of generated disjoint paths is $2n-1$. On the other hand, Algorithm S2S proposed in this paper takes $O(n^6)$ time to solve the problem, and the maximum length of generated disjoint paths is $2n-2$. Therefore, out algorithm outperforms the algorithm by Kaneko *et al.* regarding both the time complexity and the path length.

The rest of the paper is structured as follows. First, in Section II, we introduce a definition of the Möbius cube and several lemmas related to the Möbius cube. Next, in Section III, we explain each of three cases of Algorithm S2S in detail. Then, in Section IV, we prove correctness of Algorithm S2S and give its theoretical complexities. In Section V, we report the experimental results including the average performance of S2S. Finally, in Section VI, we give a conclusion and future works.

II. PRELIMINARIES

This section introduces a definition of the Möbius cube and several related lemmas.

Definition 1: An n-dimensional Möbius cube MC_n $(n \geq 0)$ is an undirected graph whose vertex set is given by $\{0, 1\}^n$. For any pair of vertices $\boldsymbol{u} = (u_1, u_2, \ldots, u_n)$ and $\boldsymbol{v}$ in MC_n, an edge $(\boldsymbol{u}, \boldsymbol{v})$ exists between them if and only if the following condition holds:

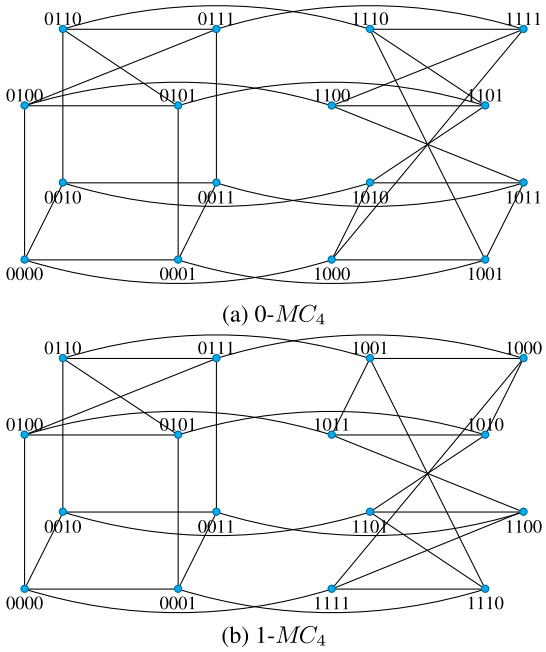
$$\boldsymbol{v} = \begin{cases} (u_1, u_2, \ldots, u_{i-1}, \bar{u}_i, u_{i+1}, \ldots, u_n) & \text{if } u_{i-1} = 0, \\ (u_1, u_2, \ldots, u_{i-1}, \bar{u}_i, \bar{u}_{i+1}, \ldots, \bar{u}_n) & \text{if } u_{i-1} = 1 \end{cases}$$

where we can assume that $u_0 = 0$ or $u_0 = 1$. For the former case, we call MC_n as an n-dimensional 0-Möbius cube 0-MC_n while for the latter, an n-dimensional 1-Möbius cube 1-MC_n. Note that MC_0 consists of a single vertex only.

If there exists an edge between two vertices $\boldsymbol{u}$ and $\boldsymbol{v}$ by the condition in Definition 1, we say that $\boldsymbol{u}$ and $\boldsymbol{v}$ are connected by an edge of i-th dimension, and we refer $\boldsymbol{v}$ and $\boldsymbol{u}$ as $\boldsymbol{u}^{(i)}$ and $\boldsymbol{v}^{(i)}$, respectively.

For instance, 0-MC_4 and 1-MC_4 are shown in Fig. 1. From Fig. 1, we can observe that MC_n is comprised of two disjoint sub graphs MC^0 and MC^1 where MC^k $(k \in \{0, 1\})$ is induced from the vertex set $\{\boldsymbol{u} = (u_1, u_2, \ldots, u_n) \mid \boldsymbol{u} \in MC_n, u_1 = k\}$ and it is isomorphic to k-MC_{n-1}. In addition, we introduce $MC^{k_1k_2}$ $(k_1, k_2 \in \{0, 1\})$, which is induced from the vertex set $\{\boldsymbol{u} = (u_1, u_2, \ldots, u_n) \mid \boldsymbol{u} \in MC_n, u_1 = k_1, u_2 = k_2\}$, and it is isomorphic to k_2-MC_{n-2}.

For an n-dimensional 0-Möbius cube, 0-MC_n, an n-dimensional 1-Möbius cube, 1-MC_n, an n-dimensional hypercube, HC_n, an n-dimensional folded hypercube, FC_n, and an n-dimensional twisted cube [1], TC_n, Table 1 compares them regarding their numbers of vertices, degrees, diameters, and average distances. From Table 1, we can see that FC_n gives the smallest diameter. However, in the up-to-date design of a



(a) 0-MC_4

(b) 1-MC_4

FIGURE 1. Examples of 0-MC_4 and 1-MC_4.

TABLE 1. Comparison of 0-MC_n and 1-MC_n with other topologies.

	#vertices	degree	diameter	average distance
0-MC_n	2^n	n	$\lceil (n+2)/2 \rceil$	†
1-MC_n	2^n	n	$\lceil (n+1)/2 \rceil$	†
HC_n	2^n	n	n	$n/2$
FC_n	2^n	$n+1$	$\lceil n/2 \rceil$	$< n/2$ [2]
TC_n	2^n	n	$\lceil (n+1)/2 \rceil$	$\to 3n/8$ $(n \to \infty)$

†: $\leq n/3 + \lceil 1 - (-1/2)^n \rceil /9 + 1$ [15]

parallel system, it is considered to be costly to increase the degree to connect the same number of vertices [22]. Also, we can see that TC_n has a slightly smaller diameter than 0-MC_n while TC_n has a much larger average distance than 0-MC_n and 1-MC_n.

MC_n has a routing algorithm, which generates one of the shortest paths of length at most $\lceil (n+2)/2 \rceil$ between an arbitrary pair of vertices in $O(n)$ time [15]. This algorithm is referred as SP in the rest of the paper.

In implementation, we assume that every vertex can be stored in one machine word, and choosing an edge by obtaining $\boldsymbol{u}^{(i)}$ for any vertex $\boldsymbol{u}$ can be done in $O(1)$ time.

Lemma 1: In MC_n, there is not a cycle of length 3. Proof: Assume that there exists a cycle C: $\boldsymbol{u} = (u_1, u_2, \ldots, u_n) \to \boldsymbol{u}^{(i)} \to \boldsymbol{u}^{(j)} \to \boldsymbol{u}$ $(i < j)$ of length 3. Then, the i-th bit of $\boldsymbol{u}^{(i)}$ is $\bar{u}_i$ while the i-th bit of $\boldsymbol{u}^{(j)}$ is u_i from $i < j$. Now, to revert the i-th bit of $\boldsymbol{u}^{(i)}$ without changing the first to $(i-1)$-th bits, it is necessary to perform the operation of $\boldsymbol{u}^{(i,i)}$. Hence, $\boldsymbol{u}^{(j)} = \boldsymbol{u}^{(i,i)} = \boldsymbol{u}$ must hold, and it contradicts that C is a cycle of length 3. Consequently, there is not a cycle of length 3 in MC_n. □

Lemma 2: For any vertex $\boldsymbol{u}$ in MC_n, exactly one edge exists between MC^0 and MC^1 including $\boldsymbol{u}$ as one of its terminal vertices.

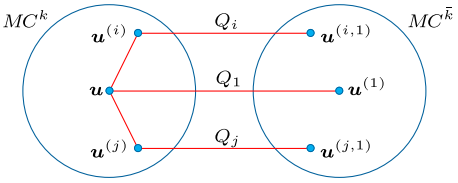


FIGURE 2. Disjoint paths between MC^0 and MC^1.

Proof: For a vertex $\boldsymbol{u}$, assume that $\boldsymbol{u} \in MC^k$ $(k \in \{0,1\})$. Then, $\boldsymbol{u}^{(i)} \in MC^k$ $(2 \leq i \leq n)$ while $\boldsymbol{u}^{(1)} \in MC^{\bar{k}}$. Hence, exactly one edge $(\boldsymbol{u}, \boldsymbol{u}^{(1)})$ exists between MC^0 and MC^1 including $\boldsymbol{u}$ as one of its terminal vertices. □

Lemma 3: For any vertex $\boldsymbol{u}$ in MC_n, there exist n paths of lengths at most 2 between MC^0 and MC^1 including $\boldsymbol{u}$ as one of their terminal vertices such that they are disjoint except for $\boldsymbol{u}$.

Proof: Let us consider n paths that include $\boldsymbol{u}$ as one of their terminal vertices and span between MC^0 and MC^1:

$$Q_i : \begin{cases} \boldsymbol{u} \to \boldsymbol{u}^{(i)} \to \boldsymbol{u}^{(i,1)} & \text{if } 2 \leq i \leq n, \\ \boldsymbol{u} \to \boldsymbol{u}^{(i)} & \text{if } i = 1 \end{cases}$$

where Q_1 is disjoint with other paths Q_i $(2 \leq i \leq n)$ except for $\boldsymbol{u}$ from Lemma 2. Moreover, for two paths Q_i and Q_j $(2 \leq i < j \leq n)$, from $\boldsymbol{u}^{(i)} \neq \boldsymbol{u}^{(j)}$ and Lemma 1, these paths are also disjoint except for $\boldsymbol{u}$ (Fig. 2). From above discussion, the n paths Q_i $(1 \leq i \leq n)$ of length at most 2 are disjoint except for $\boldsymbol{u}$. □

III. ALGORITHM S2S

We present Algorithm S2S in this section. For a source-vertex set $S = \{\boldsymbol{s}_1, \boldsymbol{s}_2, \ldots, \boldsymbol{s}_n\}$ and a target-vertex set $T = \{\boldsymbol{t}_1, \boldsymbol{t}_2, \ldots, \boldsymbol{t}_n\}$ in MC_n, the algorithm generates n disjoint paths between S and T.

A. CASE 1

Assume that $S \cup T \subset MC^k$ $(k \in \{0,1\})$ in this case. Then, the following Procedure 1 generates n disjoint paths P_i: $\boldsymbol{s}_i \rightsquigarrow \boldsymbol{t}_{j_i}$ $(1 \leq i \leq n, \{j_1, j_2, \ldots, j_n\} = \{1, 2, \ldots, n\})$.

Procedure 1:

Step 1: Apply Algorithm S2S recursively in MC^k to generate $(n-1)$ disjoint paths Q_i: $\boldsymbol{s}_i \rightsquigarrow \boldsymbol{t}_{j_i}$ $(1 \leq i \leq n-1, \{j_1, j_2, \ldots, j_{n-1}\} = \{1, 2, \ldots, n-1\})$.

Step 2: If one of Q_i $(1 \leq i \leq n-1)$, say Q_l, includes $\boldsymbol{s}_n$, delete the sub path $\boldsymbol{s}_l \rightsquigarrow \boldsymbol{s}_n$, let Q_l be $\boldsymbol{s}_n \rightsquigarrow \boldsymbol{t}_{j_l}$, and exchange the indices of $\boldsymbol{s}_l$ and $\boldsymbol{s}_n$.

Step 3: If one of Q_i $(1 \leq i \leq n-1)$, say Q_m, includes $\boldsymbol{t}_n$, delete the sub path $\boldsymbol{t}_n \rightsquigarrow \boldsymbol{t}_{j_m}$, let Q_m be $\boldsymbol{s}_m \rightsquigarrow \boldsymbol{t}_n$, and exchange the indices of $\boldsymbol{t}_{j_m}$ and $\boldsymbol{t}_n$.

Step 4: Choose edges $\boldsymbol{s}_n \to \boldsymbol{s}_n^{(1)}$ and $\boldsymbol{t}_n \to \boldsymbol{t}_n^{(1)}$.

Step 5: By using Algorithm SP, generate a path Q_n: $\boldsymbol{s}_n^{(1)} \rightsquigarrow \boldsymbol{t}_n^{(1)}$ in the $MC^{\bar{k}}$. Consequently, n disjoint paths P_i $(1 \leq i \leq n)$

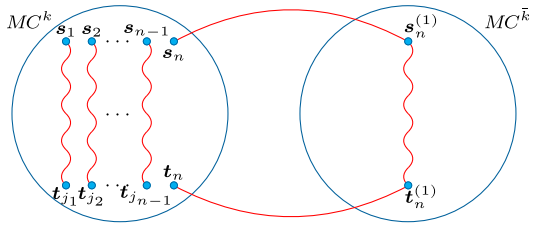


FIGURE 3. After Step 5 in Procedure 1.

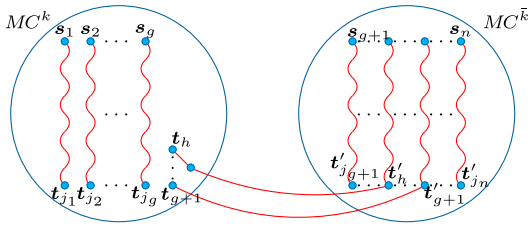


FIGURE 4. After Step 4 in Procedure 2.

are generated:

$$P_i: \begin{cases} \boldsymbol{s}_i \overset{Q_i}{\rightsquigarrow} \boldsymbol{t}_{j_i} & \text{if } 1 \le i \le n-1, \\ \boldsymbol{s}_n \to \boldsymbol{s}_n^{(1)} \overset{Q_n}{\rightsquigarrow} \boldsymbol{t}_n^{(1)} \to \boldsymbol{t}_n & \text{if } i = n. \end{cases}$$

Fig. 3 depicts the status after Step 5 in Procedure 1. Note that we use a wave line to represent a path while an arc or a line segment is used to represent an edge.

B. CASE 2

Assume that $0 < |S \cap MC^k| < n$ or $0 < |T \cap MC^{\bar{k}}| < n$ ($k \in \{0, 1\}$) in this case. Then, the following Procedure 2 generates n disjoint paths P_i: $\boldsymbol{s}_i \rightsquigarrow \boldsymbol{t}_{j_i}$ ($1 \le i \le n$, $\{j_1, j_2, \ldots, j_n\} = \{1, 2, \ldots, n\}$). Without loss of generality, we can assume that $S \cap MC^k = \{\boldsymbol{s}_1, \boldsymbol{s}_2, \ldots, \boldsymbol{s}_g\}$, $T \cap MC^k = \{\boldsymbol{t}_1, \boldsymbol{t}_2, \ldots, \boldsymbol{t}_h\}$, and $h \ge g$.

Procedure 2:

Step 1: For g source vertices $\boldsymbol{s}_1, \boldsymbol{s}_2, \ldots, \boldsymbol{s}_g$ and g target vertices $\boldsymbol{t}_1, \boldsymbol{t}_2, \ldots, \boldsymbol{t}_g$ in MC^k, apply Algorithm S2S recursively to generate g disjoint paths Q_i: $\boldsymbol{s}_i \rightsquigarrow \boldsymbol{t}_{j_i}$ ($1 \le i \le g$, $\{j_1, j_2, \ldots, j_g\} = \{1, 2, \ldots, g\}$).

Step 2: For each vertex $\boldsymbol{t}_l$ ($g+1 \le l \le h$), if one of Q_i ($1 \le i \le g$), say Q_m, includes $\boldsymbol{t}_l$, delete the sub path $\boldsymbol{t}_l \rightsquigarrow \boldsymbol{t}_{j_m}$, let Q_m be $\boldsymbol{s}_m \rightsquigarrow \boldsymbol{t}_l$, and exchange the indices of $\boldsymbol{t}_{j_m}$ and $\boldsymbol{t}_l$.

Step 3: For every vertex $\boldsymbol{t}_i$ ($g+1 \le i \le h$), do the following. First, generate n paths $R_{i,l}$ ($1 \le l \le n$) from Lemma 3. If among $R_{i,l}$ ($1 \le l \le n$) exists a path $\hat{R}_i$: $\boldsymbol{t}_i \rightsquigarrow \boldsymbol{t}'_i (\in MC^{\bar{k}})$ that does not include any vertex of Q_i ($1 \le i \le g$), any vertex of $\hat{R}_{i'}$ ($g+1 \le i' < i$), or any one of other target vertices, choose it. Otherwise, find one of Q_i ($1 \le i \le g$), say Q_x, that includes multiple vertices of $R_{i,l}$ ($1 \le l \le n$). Then, find the vertex $\boldsymbol{t}_i^{(y)}$ that is closest to $\boldsymbol{s}_x$ along Q_x, delete the sub path $\boldsymbol{t}_i^{(y)} \rightsquigarrow \boldsymbol{t}_{j_x}$, let Q_x be $\boldsymbol{s}_x \rightsquigarrow \boldsymbol{t}_i^{(y)} \to \boldsymbol{t}_i$, and exchange the indices of $\boldsymbol{t}_i$ and $\boldsymbol{t}_{j_x}$. Repeat this process until we can find $(h-g)$ paths $\hat{R}_i$ ($g+1 \le i \le h$).

Step 4: Let $\boldsymbol{t}'_i$ ($h+1 \le i \le n$) be $\boldsymbol{t}_i$. Then, for $(n-g)$ source vertices $\boldsymbol{s}_{g+1}, \boldsymbol{s}_{g+2}, \ldots, \boldsymbol{s}_n$ and $(n-g)$ target vertices $\boldsymbol{t}'_{g+1}, \boldsymbol{t}'_{g+2}, \ldots, \boldsymbol{t}'_n$ in $MC^{\bar{k}}$, apply Algorithm S2S recursively to generate $(n-g)$ disjoint paths Q_i: $\boldsymbol{s}_i \rightsquigarrow \boldsymbol{t}'_{j_i}$ ($g+1 \le i \le n$, $\{j_{g+1}, j_{g+2}, \ldots, j_n\} = \{g+1, g+2, \ldots, n\}$). Consequently, n disjoint paths P_i ($1 \le i \le n$) are generated:

$$P_i: \begin{cases} \boldsymbol{s}_i \overset{Q_i}{\rightsquigarrow} \boldsymbol{t}_{j_i} & \text{if } 1 \le j_i \le g, \\ \boldsymbol{s}_i \overset{Q_i}{\rightsquigarrow} \boldsymbol{t}'_{j_i} \overset{\hat{R}_{j_i}}{\rightsquigarrow} \boldsymbol{t}_{j_i} & \text{if } g+1 \le j_i \le h, \\ \boldsymbol{s}_i \overset{Q_i}{\rightsquigarrow} \boldsymbol{t}'_{j_i} (= \boldsymbol{t}_{j_i}) & \text{if } h+1 \le j_i \le n. \end{cases}$$

Fig. 4 depicts the configuration after Step 4 in Procedure 2.

C. CASE 3

Assume that $S \subset MC^k$ and $T \subset MC^{\bar{k}}$ in this case. Then, the following procedures generate n disjoint paths P_i: $\boldsymbol{s}_i \rightsquigarrow \boldsymbol{t}_{j_i}$ ($1 \le i \le n$, $\{j_1, j_2, \ldots, j_n\} = \{1, 2, \ldots, n\}$) by considering four sub Möbius cubes: $MC^{k_1 k_2}$, $MC^{k_1 \bar{k}_2}$, $MC^{\bar{k}_1 k_3}$, and $MC^{\bar{k}_1 \bar{k}_3}$ where $k_1, k_2 \in \{0, 1\}$ and $k_3 = k_2$ in 0-MC_n, and $k_3 = \bar{k}_2$ in 1-MC_n. When $n = 3$, in every sub case, it is necessary to generate $2 (= \lceil n/2 \rceil)$ disjoint paths in one of the four 1-dimensional sub Möbius cubes, which has only two vertices. However, because two source vertices (and two target vertices) are distributed to distinct vertices in the sub Möbuis cube, every vertex in the sub Möbius cube has a pair of the source and target vertices. Hence, two disjoint paths can be generated because every pair is initially connected by a path of length 0. In the rest of this case, we can assume without loss of generality that $|S \cap MC^{k_1 k_2}| \ge |S \cap MC^{k_1 \bar{k}_2}|$.

1) CASE 3-1

First, we assume that $|T \cap MC^{\bar{k}_1 k_3}| \ge |T \cap MC^{\bar{k}_1 \bar{k}_3}|$. Without loss of generality, we can assume that $S \cap MC^{k_1 k_2} = \{\boldsymbol{s}_1, \boldsymbol{s}_2, \ldots, \boldsymbol{s}_{n_s}\}$ and $T \cap MC^{\bar{k}_1 k_3} = \{\boldsymbol{t}_1, \boldsymbol{t}_2, \ldots, \boldsymbol{t}_{n_t}\}$. Note that $n_s \ge n - n_s$ and $n_t \ge n - n_t$. Then, we generate n disjoint paths P_i: $\boldsymbol{s}_i \rightsquigarrow \boldsymbol{t}_{j_i}$ ($1 \le i \le n$, $\{j_1, j_2, \ldots, j_n\} = \{1, 2, \ldots, n\}$) by the following Procedure 3.

Procedure 3:

Step 1: Find $(n_s - \lceil n/2 \rceil)$ vertices $\boldsymbol{s}_i$ in $S \cap MC^{k_1 k_2}$ such that $\boldsymbol{s}_i^{(2)} \notin S$. We can assume without loss of generality that $\{\boldsymbol{s}^{(2)}_{\lceil n/2 \rceil + 1}, \boldsymbol{s}^{(2)}_{\lceil n/2 \rceil + 2}, \ldots, \boldsymbol{s}^{(2)}_{n_s}\} \cap S = \emptyset$.

Step 2: Choose $\lceil n/2 \rceil$ edges $\boldsymbol{s}_i \to \boldsymbol{s}_i^{(1)}$ ($1 \le i \le \lceil n/2 \rceil$).

Step 3: For $\lceil n/2 \rceil$ vertices $\boldsymbol{s}_1^{(1)}, \boldsymbol{s}_2^{(1)}, \ldots, \boldsymbol{s}^{(1)}_{\lceil n/2 \rceil}$ and $\lceil n/2 \rceil$ vertices $\boldsymbol{t}_1, \boldsymbol{t}_2, \ldots, \boldsymbol{t}_{\lceil n/2 \rceil}$, apply Algorithm S2S recursively in $MC^{\bar{k}_1 k_3}$ to generate $\lceil n/2 \rceil$ disjoint paths Q_i: $\boldsymbol{s}_i^{(1)} \rightsquigarrow \boldsymbol{t}_{j_i}$ ($1 \le i \le \lceil n/2 \rceil$, $\{j_1, j_2, \ldots, j_{\lceil n/2 \rceil}\} = \{1, 2, \ldots, \lceil n/2 \rceil\}$).

Step 4: For each vertex $\boldsymbol{t}_l$ ($\lceil n/2 \rceil + 1 \le l \le n_t$), if one of Q_i ($1 \le i \le \lceil n/2 \rceil$), say Q_m, includes $\boldsymbol{t}_l$, delete the sub path $\boldsymbol{t}_l \rightsquigarrow \boldsymbol{t}_{j_m}$, let Q_m be $\boldsymbol{s}_m \rightsquigarrow \boldsymbol{t}_l$, and exchange the indices of $\boldsymbol{t}_{j_m}$ and $\boldsymbol{t}_l$.

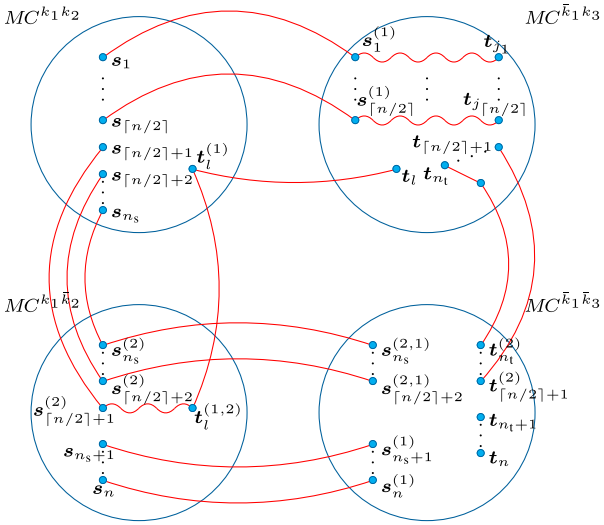


FIGURE 5. After Step 7 in Procedure 3 (in case where t_l was found, and $t_l^{(1)} \notin S \cap MC^{k_1k_2}$ in Step 5).

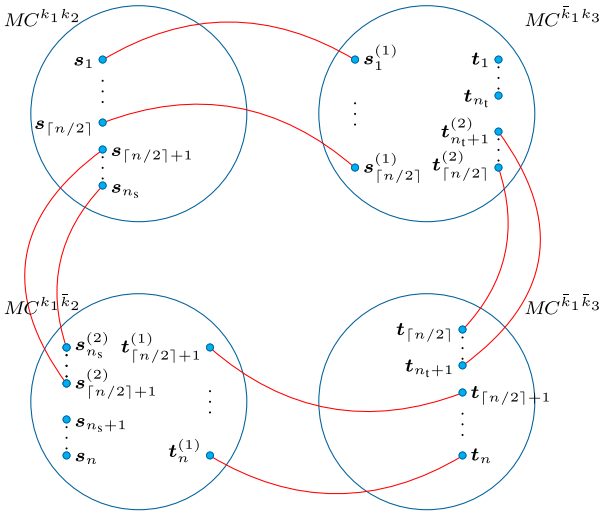


FIGURE 6. After Step 6 in Procedure 4.

Step 5: For $\boldsymbol{t}_i$ ($\lceil n/2 \rceil + 1 \le i \le n_t$), try to generate disjoint paths $\boldsymbol{t}_i \rightsquigarrow \boldsymbol{t}'_i (\in MC^{\bar{k}_1\bar{k}_3})$ of lengths at most 2 as in Step 3 of Case 2. If there is a vertex, say $\boldsymbol{t}_l$, from which a disjoint path cannot be generated, choose an edge $\boldsymbol{t}_l \to \boldsymbol{t}_l^{(1)}$. If $\boldsymbol{t}_l^{(1)} \in S \cap MC^{k_1k_2}$, we can assume without loss of generality that $\boldsymbol{t}_l^{(1)} = \boldsymbol{s}_{\lceil n/2 \rceil+1}$. Otherwise, that is, if $\boldsymbol{t}_l^{(1)} \notin S \cap MC^{k_1k_2}$, choose the edges $\boldsymbol{t}_l^{(1)} \to \boldsymbol{t}_l^{(1,2)}$ and $\boldsymbol{s}_{\lceil n/2 \rceil+1} \to \boldsymbol{s}^{(2)}_{\lceil n/2 \rceil+1} (= \boldsymbol{s}'_{\lceil n/2 \rceil+1})$.

Step 6: Choose $(n_s - \lceil n/2 \rceil - 1)$ edges $\boldsymbol{s}_i \to \boldsymbol{s}_i^{(2)} (= \boldsymbol{s}'_i)$ ($\lceil n/2 \rceil + 2 \le i \le n_s$). Let $\boldsymbol{s}_i$ be $\boldsymbol{s}'_i$ ($n_s + 1 \le i \le n$). If the vertex $\boldsymbol{t}_l$ was not found in Step 5, choose the edge $\boldsymbol{s}_{\lceil n/2 \rceil+1} \to \boldsymbol{s}^{(2)}_{\lceil n/2 \rceil+1} (= \boldsymbol{s}'_{\lceil n/2 \rceil+1})$.

Step 7: If the edge $\boldsymbol{t}_l^{(1)} \to \boldsymbol{t}_l^{(1,2)}$ was not chosen in Step 5, choose edges $\boldsymbol{s}'_i \to \boldsymbol{s}_i^{\prime(1)}$ ($\lceil n/2 \rceil + 2 \le i \le n$), and go to Step 8. Otherwise, if the edge $\boldsymbol{t}_l^{(1)} \to \boldsymbol{t}_l^{(1,2)}$ was chosen in Step 5, apply Algorithm SP in $MC^{k_1\bar{k}_2}$ to generate one of the shortest paths between $\boldsymbol{s}'_{\lceil n/2 \rceil+1}$ and $\boldsymbol{t}_l^{(1,2)}$. If the path includes some of the vertices $\boldsymbol{s}'_{\lceil n/2 \rceil+2}, \boldsymbol{s}'_{\lceil n/2 \rceil+3}, \ldots, \boldsymbol{s}'_n$, choose one of them, say $\boldsymbol{s}'_y$, that is closest to $\boldsymbol{t}_l^{(1,2)}$ along the path, delete the sub path $\boldsymbol{s}'_{\lceil n/2 \rceil+1} \rightsquigarrow \boldsymbol{s}'_y$, exchange the indices of $\boldsymbol{s}'_y$ and $\boldsymbol{s}'_{\lceil n/2 \rceil+1}$. Choose edges $\boldsymbol{s}'_i \to \boldsymbol{s}_i^{\prime(1)}$ ($\lceil n/2 \rceil + 2 \le i \le n$). See Figure 5.

Step 8: Let $\boldsymbol{t}_i = \boldsymbol{t}'_i$ ($n_{t+1} \le i \le n$). If the vertex $\boldsymbol{t}_l$ was not found in Step 5, apply Algorithm S2S recursively in $MC^{\bar{k}_1\bar{k}_3}$ to generate $\lfloor n/2 \rfloor$ disjoint paths Q_i: $\boldsymbol{s}_i^{\prime(1)} \rightsquigarrow \boldsymbol{t}'_{j_i}$ ($\lceil n/2 \rceil + 1 \le i \le n$, $\{j_{\lceil n/2 \rceil+1}, j_{\lceil n/2 \rceil+2}, \ldots, j_n\} = \{\lceil n/2 \rceil + 1, \lceil n/2 \rceil + 2, \ldots, n\}$). Otherwise, if $\boldsymbol{t}_l$ was found in Step 5, apply Algorithm S2S recursively in $MC^{\bar{k}_1\bar{k}_3}$ to generate $(\lfloor n/2 \rfloor - 1)$ disjoint paths Q_i: $\boldsymbol{s}_i^{\prime(1)} \rightsquigarrow \boldsymbol{t}'_{j_i}$ ($\lceil n/2 \rceil + 2 \le i \le n$, $\{j_{\lceil n/2 \rceil+2}, j_{\lceil n/2 \rceil+3}, \ldots, j_n\} = \{\lceil n/2 \rceil + 1, \lceil n/2 \rceil + 2, \ldots, l-1, l+1, \ldots, n\}$).

2) CASE 3-2

Now, we assume that $|T \cap MC^{\bar{k}_1k_3}| < |T \cap MC^{\bar{k}_1\bar{k}_3}|$. Without loss of generality, we can assume that $S \cap MC^{k_1k_2} = \{\boldsymbol{s}_1, \boldsymbol{s}_2, \ldots, \boldsymbol{s}_{n_s}\}$ and $T \cap MC^{\bar{k}_1k_3} = \{\boldsymbol{t}_1, \boldsymbol{t}_2, \ldots, \boldsymbol{t}_{n_t}\}$. Note that $n_s \ge n - n_s$ and $n_t < n - n_t$. Then, we generate n disjoint paths P_i: $\boldsymbol{s}_i \rightsquigarrow \boldsymbol{t}_{j_i}$ ($1 \le i \le n$, $\{j_1, j_2, \ldots, j_n\} = \{1, 2, \ldots, n\}$) by following Procedure 4.

Procedure 4:

Step 1: Find $(n_s - \lceil n/2 \rceil)$ vertices $\boldsymbol{s}_i$ in $S \cap MC^{k_1k_2}$ such that $\boldsymbol{s}_i^{(2)} \notin S$. We can assume without loss of generality that $\{\boldsymbol{s}^{(2)}_{\lceil n/2 \rceil+1}, \boldsymbol{s}^{(2)}_{\lceil n/2 \rceil+2}, \ldots, \boldsymbol{s}^{(2)}_{n_s}\} \cap S = \emptyset$.

Step 2: Choose $\lceil n/2 \rceil$ edges $\boldsymbol{s}_i \to \boldsymbol{s}_i^{(1)}$ ($1 \le i \le \lceil n/2 \rceil$).

Step 3: Choose $(n_s - \lceil n/2 \rceil)$ edges $\boldsymbol{s}_i \to \boldsymbol{s}_i^{(2)} (= \boldsymbol{s}'_i)$ ($\lceil n/2 \rceil + 1 \le i \le n_s$). Let $\boldsymbol{s}'_i$ be $\boldsymbol{s}_i$ ($n_s + 1 \le i \le n$).

Step 4: Choose $(\lceil n/2 \rceil - n_t)$ vertices $\boldsymbol{t}_i$ in $T \cap MC^{\bar{k}_1\bar{k}_3}$ such that $\boldsymbol{t}_i^{(2)} \notin T$. We can assume without loss of generality that $\{\boldsymbol{t}^{(2)}_{n_t+1}, \boldsymbol{t}^{(2)}_{n_t+2}, \ldots, \boldsymbol{t}^{(2)}_{\lceil n/2 \rceil}\} \cap T = \emptyset$.

Step 5: Choose $\lfloor n/2 \rfloor$ edges $\boldsymbol{t}_i \to \boldsymbol{t}_i^{(1)}$ ($\lceil n/2 \rceil + 1 \le i \le n$).

Step 6: Choose $(\lceil n/2 \rceil - n_t)$ edges $\boldsymbol{t}_i \to \boldsymbol{t}_i^{(2)} (= \boldsymbol{t}'_i)$ ($n_t + 1 \le i \le \lceil n/2 \rceil$). Let $\boldsymbol{t}'_i$ be $\boldsymbol{t}_i$ ($1 \le i \le n_t$). See Fig. 6.

Step 7: Apply Algorithm S2S recursively in $MC^{\bar{k}_1k_3}$ to generate $\lceil n/2 \rceil$ disjoint paths Q_i: $\boldsymbol{s}_i^{(1)} \rightsquigarrow \boldsymbol{t}'_{j_i}$ ($1 \le i \le \lceil n/2 \rceil$, $\{j_1, j_2, \ldots, j_{\lceil n/2 \rceil}\} = \{1, 2, \ldots, \lceil n/2 \rceil\}$).

Step 8: Apply Algorithm S2S recursively in $MC^{k_1\bar{k}_2}$ to generate $\lfloor n/2 \rfloor$ disjoint paths Q_i: $\boldsymbol{s}'_i \rightsquigarrow \boldsymbol{t}_{j_i}^{(1)}$ ($\lceil n/2 \rceil + 1 \le i \le n$, $\{j_{\lceil n/2 \rceil+1}, j_{\lceil n/2 \rceil+2}, \ldots, j_n\} = \{\lceil n/2 \rceil + 1, \lceil n/2 \rceil + 2, \ldots, n\}$).

IV. PROOF OF CORRECTNESS AND ESTIMATION OF COMPLEXITIES

In this section, we prove the correctness of our algorithm and we give the estimates of time complexity $\tau(n, g)$ of our algorithm to generate g disjoint paths in an n-dimensional Möbius cube, and maximum length $\lambda(n)$ of the paths generated by our algorithm. Proof is based on induction on n.

Lemma 4: Procedure 1 generates n disjoint paths in MC_n in $\tau(n-1, n-1) + O(n\lambda(n-1))$ time. The lengths of the generated paths are at most $\max\{\lambda(n-1), \lfloor n/2 \rfloor + 3\}$.

Proof: The $(n-1)$ paths Q_i ($1 \le i \le n-1$) generated in Steps 1, 2, and 3 are disjoint from one another from the

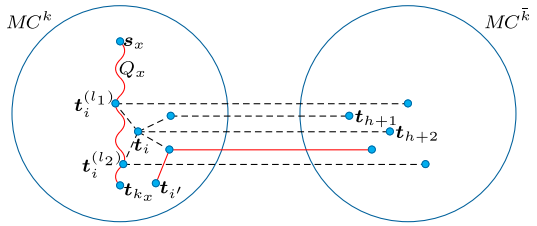


FIGURE 7. Replacement of Q_x in Lemma 5.

hypothesis of induction. Also, these paths are disjoint from s_n and t_n. The path generated in Steps 4 and 5 is included in $MC^{\bar{k}}$ except for s_n and t_n. Hence, the path is disjoint from the paths generated in Steps 1, 2, and 3.

Step 1 takes $\tau(n-1, n-1)$ time to generate $(n-1)$ paths of lengths at most $\lambda(n-1)$ from the hypothesis of induction. Step 2 takes $O(n\lambda(n-1))$ time to check if s_n is included in Q_i $(1 \le i \le n-1)$ or not. It takes $O(1)$ time to delete the sub path, update Q_l, and exchange the indices. Similarly, Step 3 takes $O(n\lambda(n-1))$ time to check if t_n is included in Q_i $(1 \le i \le n-1)$ or not. It takes $O(1)$ time to delete the sub path, update Q_m, and exchange the indices. Step 4 takes $O(1)$ time to choose two edges. In Step 5, Algorithm SP takes $O(n)$ time to generate the path of length at most $\lceil (n+1)/2 \rceil$.

Hence, Procedure 1 takes $\tau(n-1, n-1) + O(n\lambda(n-1))$ time to generate n disjoint paths of lengths at most $\max\{\lambda(n-1), \lfloor n/2 \rfloor + 3\}$. Note that $2 + \lceil (n+1)/2 \rceil = \lfloor n/2 \rfloor + 3$. □

Lemma 5: In Step 3 of Procedure 2, for every vertex t_i $(g+1 \le i \le h)$, we can find a path $\hat{R}_i$.

Proof: There are n candidate paths $R_{i,l}$ $(1 \le l \le n)$. The path $\hat{R}_{i'}$ for a vertex $t_{i'}$ can block at most one of the n candidate paths $R_{i,l}$ from Lemma 1. In addition, each of the vertices $t_{h+1}, t_{h+2}, \ldots, t_n$ can block at most one of the n candidate paths $R_{i,l}$. Therefore, if all of the n candidate paths are blocked, at least one of the g disjoint paths Q_i: $s_i \rightsquigarrow t_{k_i}$ $(1 \le i \le g)$, say Q_x, must block multiple candidate paths. Assume that the paths R_{i,l_1}: $t_i \to t_i^{(l_1)} \to t_i^{(l_1,1)}$ and R_{i,l_2}: $t_i \to t_i^{(l_2)} \to t_i^{(l_2,1)}$ are blocked by Q_x (Fig. 7). Then, from Lemma 1, it never happens that Q_x: $s_x \rightsquigarrow t_i^{(l_1)} \to t_{k_x} (= t_i^{(l_2)})$. Hence, the path $s_x \rightsquigarrow t_i^{(l_1)} \to t_i$ is strictly shorter than Q_x. Therefore, if we replace Q_x by the path $s_x \rightsquigarrow t_i^{(l_1)} \to t_i$ and repeat this process, we can find a path $\hat{R}_i$. The total number of replacements is restricted by the total lengths of paths Q_i. □

Lemma 6: Procedure 2 generates n disjoint paths in MC_n in $\tau(n-1, g) + \tau(n-1, n-g) + O(n^3\{\lambda(n-1)\}^2)$ time. The lengths of the generated paths are at most $\lambda(n-1)+2$.

Proof: The g paths Q_i $(1 \le i \le g)$ generated in Steps 1 and 2 are disjoint from one another from the hypothesis of induction. The $(h-g)$ paths generated in Step 3 are disjoint from one another, and they are also disjoint from the paths generated in Steps 1 and 2. The $(n-g)$ paths generated in Step 4 are disjoint from one another from the hypothesis of induction. Moreover, because these paths are included in $MC^{\bar{k}}$, they are disjoint from the g paths generated in Steps 1 and 2, and disjoint from the $(h-g)$ paths generated in Step 3 except for $t'_{g+1}, t'_{g+2}, \ldots, t'_h$.

Step 1 takes $\tau(n-1, g)$ time to generate g paths of lengths at most $\lambda(n-1)$ from the hypothesis of induction. Step 2 takes $O(n^2\lambda(n-1))$ time to check if t_l $(g+1 \le l \le h)$ are included in Q_i $(1 \le i \le g)$ or not. It takes $O(n)$ time to delete the sub paths, update Q_m's, and exchange the indices. Step 3 takes $O(n^2\lambda(n-1))$ time to check if there is an available path $R_{i,l}$ or not. If there is not any available path, it takes $O(\lambda(n-1))$ time to find $t_i^{(y)}$. Note that it is possible to find Q_x while checking the existence of an available path. It takes $O(1)$ time to delete the sub path, update Q_x, and exchange the indices. This check followed by the process is repeated at most $O(n\lambda(n-1))$ times from Lemma 5. Hence, it takes $O(n^3\{\lambda(n-1)\}^2)$ time in Step 3 to find $(h-g)$ paths $\hat{R}_i$ $(g+1 \le i \le h)$ of lengths at most 2. Step 4 takes $\tau(n-1, n-g)$ time to generated $(n-g)$ paths of lengths at most $\lambda(n-1)$ from the hypothesis of induction.

Hence, in total, Procedure 2 takes $\tau(n-1, g) + \tau(n-1, n-g) + O(n^3\{\lambda(n-1)\}^2)$ time to generate n disjoint paths of lengths at most $\lambda(n-1)+2$. □

Lemma 7: In Step 5 of Procedure 3, there is at most one vertex, say t_l, from which any disjoint path cannot be generated.

Proof: t_l exists if and only if each of the $(n-1)$ disjoint paths from t_l to $MC^{\bar{k}_1\bar{k}_3}$ is blocked by the paths generated in Steps 4 and 5, or by the target vertices in $MC^{\bar{k}_1 k_3}$ and $MC^{\bar{k}_1\bar{k}_3}$. Note that each of the paths generated in Steps 4 and 5 can block at most one of the $(n-1)$ paths from t_l after repeating the process as in Step 3 of Case 2. Then, because of the edge $t_l \to t_l^{(2)}$, $t_l^{(2)}$ is one of the target vertices in $MC^{\bar{k}_1\bar{k}_3}$. Hence, there is a disjoint path from each of other target vertices t_i $(\lceil n/2 \rceil + 1 \le i \ne l \le n_t)$ to $MC^{\bar{k}_1\bar{k}_3}$ because t_l and $t_l^{(2)}$ cannot block two distinct paths from the vertex t_i. □

Lemma 8: Procedure 3 generates n disjoint paths in MC_n in $\tau(n-2, \lceil n/2 \rceil) + \tau(n-2, \lfloor n/2 \rfloor) + O(n^3\{\lambda(n-2)\}^2)$ time. The lengths of the generated paths are at most $\max\{\lambda(n-2)+4, \lceil n/2 \rceil + 3\}$.

Proof: $\lceil n/2 \rceil$ edges chosen in Step 2 are disjoint from one another from Lemma 2. They are also disjoint from other source vertices from $S \cap MC^{\bar{k}_1 k_3} = \emptyset$. $\lceil n/2 \rceil$ paths generated in Steps 3 and 4 are disjoint from one another from the hypothesis of induction. They are also disjoint from the edges chosen in Step 2 except for $s_i^{(1)}$ $(1 \le i \le \lceil n/2 \rceil)$. Moreover, the paths are disjoint from other target vertices. The paths $t_i \rightsquigarrow t'_i$ of lengths at most 2 generated in Step 5 are disjoint from one another. They are also disjoint from the edges chosen in Step 2 and the paths generated in Steps 3 and 4. If either of the paths $t_l \to t_l^{(1)} (= s_{\lceil n/2 \rceil + 1})$ or $t_l \to t_l^{(1)} \to t_l^{(1,2)}$ is generated, it is disjoint from other paths generated so far. Also, it is disjoint from other target vertices. Moreover, if the edge $s_{\lceil n/2 \rceil + 1} \to s_{\lceil n/2 \rceil + 1}^{(2)}$ is chosen, it is disjoint from other source vertices because $s_{\lceil n/2 \rceil + 1}$ is chosen such that $s_{\lceil n/2 \rceil + 1}^{(2)} \notin S$ in Step 1. $(n_s - \lceil n/2 \rceil - 1)$ edges $s_i \to s_i^{(2)}$ $(\lceil n/2 \rceil + 2 \le i \le n_s)$ chosen in Step 6

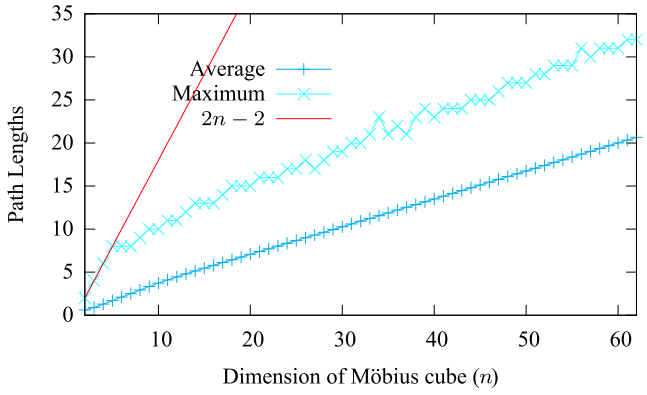


FIGURE 8. Length of paths generated by Algorithm S2S in 0-MC_n.

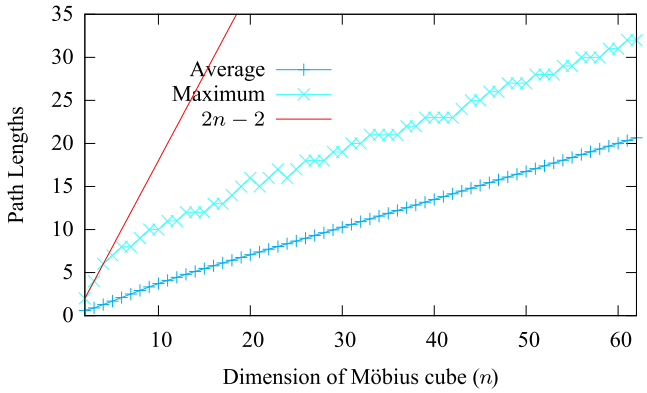


FIGURE 9. Length of paths generated by Algorithm S2S in 1-MC_n.

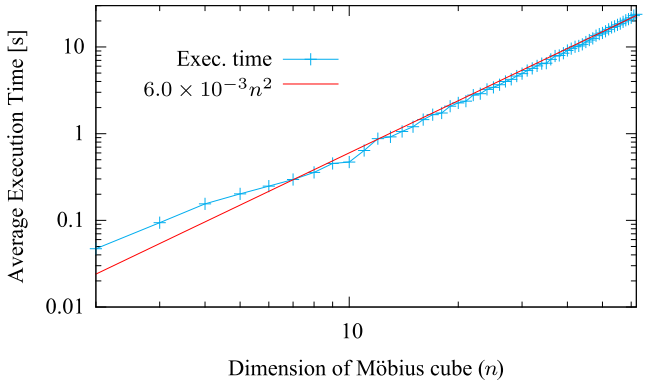


FIGURE 10. Average execution time of Algorithm S2S in 0-MC_n.

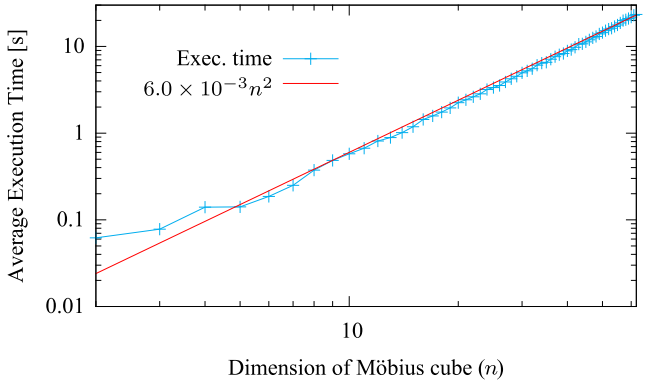


FIGURE 11. Average execution time of Algorithm S2S in 1-MC_n.

are disjoint from one another from Lemma 2. Also, these edges are disjoint from other source vertices because s_i were chosen such that $s_i^{(2)} \notin S$ in Step 1. In Step 7, if the path $s'_{\lceil n/2\rceil+1} \rightsquigarrow t_l^{(1,2)}$ is generated by Algorithm SP, it is disjoint from other paths except for $s'_{\lceil n/2\rceil+1}$ and $t_l^{(1,2)}$ because it is obtained by excluding the vertices s'_i ($\lceil n/2\rceil + 2 \le i \le n$). Also, ($\lfloor n/2\rfloor - 1$) edges $s'_i \rightarrow s_i^{\prime(1)}$ chosen in Step 7 are disjoint from one another from Lemma 2. In addition, from $S \cap MC^{\bar{k}_1\bar{k}_3} = \emptyset$, the edges are disjoint from other source vertices. $\lfloor n/2\rfloor$ (or ($\lfloor n/2\rfloor - 1$)) paths generated in Step 8 are disjoint from one another from the hypothesis of induction. Because these paths are all included in $MC^{\bar{k}_1\bar{k}_3}$, they are disjoint from other paths generated in Steps 2 to 7 except for some of their terminal vertices.

It takes $O(n^2)$ time to find ($n_s - \lceil n/2\rceil$) vertices $s_i (\in S \cap MC^{k_1k_2})$ such that $s_i^{(2)} \notin S$ in Step 1. Step 2 takes $O(n)$ time to choose $\lceil n/2\rceil$ edges. Step 3 takes $\tau(n-2, \lceil n/2\rceil)$ time to generate $\lceil n/2\rceil$ paths of lengths at most $\lambda(n-2)$ from the hypothesis of induction. In Step 4, it takes $O(n^2\lambda(n-2))$ time to check if t_l ($\lceil n/2\rceil + 1 \le l \le n_t$) are included in Q_i ($1 \le i \le \lceil n/2\rceil$) or not, and it takes $O(n)$ time to delete the sub paths, update Q_m's, and exchange the indices. From the proof of Lemma 6, Step 5 takes $O(n^3\{\lambda(n-2)\}^2)$ time to generate paths $t_i \rightsquigarrow t'_i$ paths of lengths at most 2 in Step 5. It takes $O(1)$ time to choose the edge $t_l \rightarrow t_l^{(1)}$. It takes $O(n)$ time to check if $t_l^{(1)} \in S \cap M^{k_1k_2}$ or not. It takes $O(1)$ time to choose the edges $t_l^{(1)} \rightarrow t_l^{(1,2)}$ and $s_{\lceil n/2\rceil+1} \rightarrow s^{(2)}_{\lceil n/2\rceil+1}$. Step 6 takes $O(n)$ time to choose ($n_s - \lceil n/2\rceil - 1$) edges. In Step 7, Algorithm SP takes $O(n)$ time to generate the path $s'_{\lceil n/2\rceil+1} \rightsquigarrow t_l^{(1,2)}$ of length at most $\lceil n/2\rceil$. Also, it takes $O(n^2)$ time to update this path such that it does not include the vertices $s'_{\lceil n/2\rceil+2}, s'_{\lceil n/2\rceil+3}, \ldots, s'_n$ It takes $O(n)$ time to choose ($\lfloor n/2\rfloor - 1$) edges. Step 8 takes $\tau(n-2, \lfloor n/2\rfloor)$ (or $\tau(n-2, \lfloor n/2\rfloor - 1)$) time to generate $\lfloor n/2\rfloor$ (or ($\lfloor n/2\rfloor - 1$)) paths of lengths at most $\lambda(n-2)$ from the hypothesis of induction.

Hence, in total, it takes $\tau(n-2, \lceil n/2\rceil) + \tau(n-2, \lfloor n/2\rfloor) + O(n^3\{\lambda(n-2)\}^2)$ time to generate n disjoint paths $s_i \rightsquigarrow t_{k_i}$ ($1 \le i \le n$, $\{k_1, k_2, \ldots, k_n\} = \{1, 2, \ldots, n\}$) whose lengths are at most $\max\{\lambda(n-2)+4, \lceil n/2\rceil + 3\}$. □

Lemma 9: Procedure 4 generates n disjoint paths in MC_n in $\tau(n-2, \lceil n/2\rceil) + \tau(n-2, \lfloor n/2\rfloor) + O(n^2)$ time. The lengths of the generated paths are at most $\lambda(n-2)+2$.

Proof: The $\lceil n/2\rceil$ edges chosen in Step 2 are disjoint from one another from Lemma 2. Also, from $S \cap MC^{\bar{k}_1\bar{k}_3} = \emptyset$, they are disjoint from other source vertices. The ($n_s - \lceil n/2\rceil$) edges $s_i \rightarrow s_i^{(2)}$ ($\lceil n/2\rceil + 1 \le i \le n_s$) chosen in Step 3 are also disjoint from one another from Lemma 2. Moreover, they are disjoint from other source vertices because s_i were

```
function SP(s, t)
begin
  u(= (u_1, u_2, ..., u_n)) := s ⊕ t; S := ∅; i := 1;
  while i <= n do
    if u_i = 0 then i := i + 1
    else if i = n then begin S := S ∪ {E_i}; i := i + 1 end
    else if u_{i+1} = 0 then begin S := S ∪ {e_i}; i := i + 2 end
    else begin S := S ∪ {E_i}; u := ū; i := i + 2 end;
  return SP0(s, t, S)
end;

function SP0(s, t, S)
begin
  if S = ∅ return [s];
  i := min{i | e_i ∈ S or E_i ∈ S};
  if e_i ∈ S and s_{i-1} = 0 then
    if (e_{i+1} ∈ S and s_i = 0) or (E_{i+1} ∈ S and s_i = 1) then
      return (SP0(s, t ⊕ e_i, S \ {e_i}) ++ [t])
    else return ([s] ++ SP0(s ⊕ e_i, t, S \ {e_i}))
  else if E_i ∈ S and s_{i-1} = 1 then begin
    P := [s];
    for j := i + 1 to n do
      if e_j ∈ S and s_{j-1} = 0 then
        begin P := P ++ [s ⊕ e_j]; s := s ⊕ e_j; S := S \ {e_j} end
      else if E_j ∈ S and s_{j-1} = 1 then
        begin P := P ++ [s ⊕ E_j]; s := s ⊕ E_j; S := S \ {E_j} end;
    return (P ++ SP0(s ⊕ E_i, t, S \ {E_i}))
  end
  else if e_i ∈ S and s_{i-1} = 1 then
    if {e_i, e_{i+2}, ..., e_{i+2k-2}, E_{i+2k}} ⊂ S (k ≥ 0) then
      return SP0(s, t, (S ∪ {E_i, e_{i+1}, e_{i+3}, ..., e_{i+2k-1}}) \ {e_i, e_{i+2}, ..., e_{i+2k-2}, E_{i+2k}})
    else return SP0(s, t, (S ∪ {E_i, E_{i+1}}) \ {e_i})
  else /* E_i ∈ S and s_{i-1} = 0 */
    if {E_i, e_{i+2}, e_{i+4}, ..., e_{i+2k-2}, E_{i+2k}} ⊂ S (k ≥ 0) then
      return SP0(s, t, (S ∪ {e_i, e_{i+1}, e_{i+3}, ..., e_{i+2k-1}}) \ {E_i, e_{i+2}, ..., e_{i+2k-2}, E_{i+2k}})
    else return SP0(s, t, (S ∪ {e_i, E_{i+1}}) \ {E_i})
  end
end
```

FIGURE 12. Pseudo code of Algorithm SP.

chosen such that $s_i^{(2)} \notin S$ in Step 1. Similarly, the $\lfloor n/2 \rfloor$ edges chosen in Step 5 and the $(\lceil n/2 \rceil - n_t)$ edges chosen in Step 6 are disjoint from one another and disjoint from other target vertices. The $\lceil n/2 \rceil$ paths generated in Step 7 are disjoint from one another from the hypothesis of induction. These paths are disjoint from the edges chosen in Steps 2, 3, 5, and 6 except for some of their terminal vertices because the paths are all included in $MC^{\bar{k}_1 k_3}$. The $\lfloor n/2 \rfloor$ paths generated in Step 8 are disjoint from one another from the hypothesis of induction. These paths are disjoint from the edges chosen in Steps 2, 3, 5, and 6, and the paths generated in Step 7 except for some of their terminal vertices because the paths are all included in $MC^{k_1 \bar{k}_2}$.

In Step 1, it takes $O(n^2)$ time to find $(n_s - \lceil n/2 \rceil)$ vertices $s_i (\in MC^{k_1 k_2})$ that satisfy $s_i^{(2)} \notin S$. Step 2 takes $O(n)$ time to choose $\lceil n/2 \rceil$ edges. Step 3 takes $O(n)$ time to choose $(n_s - \lceil n/2 \rceil)$ edges. In Step 4, it takes $O(n^2)$ time to find $(\lceil n/2 \rceil - n_t)$ vertices $t_i (\in MC^{\bar{k}_1 \bar{k}_3})$ that satisfy $t_i^{(2)} \notin D$. Step 5 takes $O(n)$ time to choose $\lfloor n/2 \rfloor$ edges. Step 6 takes $O(n)$ time to choose $(\lceil n/2 \rceil - n_t)$ edges. Step 7 takes $\tau(n-2, \lceil n/2 \rceil)$ time to generate $\lceil n/2 \rceil$ paths of lengths at most $\lambda(n-2)$ from the hypothesis of induction. Step 8 takes $\tau(n-2, \lfloor n/2 \rfloor)$ time to generate $\lfloor n/2 \rfloor$ paths of lengths at most $\lambda(n-2)$ from the hypothesis of induction.

Therefore, the time complexity of Procedure 4 is $\tau(n-2, \lceil n/2 \rceil) + \tau(n-2, \lfloor n/2 \rfloor) + O(n^2)$ and the maximum length of the generated paths is $\lambda(n-2)+2$. □

Theorem 1: For a set of n vertices $S = \{s_1, s_2, \ldots, s_n\}$ and a set of n vertices $T = \{t_1, t_2, \ldots, t_n\}$ in MC_n, Algorithm S2S generates n disjoint paths P_i: $s_i \rightsquigarrow t_{k_i}$ $(1 \le i \le n, \{k_1, k_2, \ldots, k_n\} = \{1, 2, \ldots, n\})$. The time complexity $\tau(n, n)$ of S2S is $O(n^6)$, and the maximum path length $\lambda(n)$ is $2n-2$.

Proof: From Lemmas 4 to 9, the generated paths are disjoint. Also, $\lambda(n) = 2n-2$ from $\lambda(n) = \lambda(n-1)+2$ and $\lambda(2) = 2$. Then, $\tau(n, n) = O(n^6)$ from $\tau(n, n) = \tau(n-1, g) + \tau(n-1, n-g) + O(n^3\{\lambda(n-1)\}^2)$ and $\tau(n, 1) = O(n)$. □

V. COMPUTER EXPERIMENT

To evaluate the average performance of Algorithm S2S, we have conducted a computer experiment as follows.

Step 1: For each of n $(1 \le n \le 62)$, repeat Steps 2 and 3 100,000 times.

Step 2: Choose a set of n distinct vertices S and a set of n distinct vertices T in 0-MC_n and 1-MC_n. Note that it is allowed to make S and T overlapped ($S \cap T \neq \emptyset$).

Step 3: Apply S2S to S and T, and measure the execution time and the path lengths.

Figs. 8 and 9 show the lengths of the paths generated by our algorithm in 0-MC_n and 1-MC_n, respectively. From these figures, we can observe that there is not a significant difference between 0-MC_n and 1-MC_n. Also, the theoretical upper bound of the maximum path length is attained with only small n, and there is a large gap between the upper bound and the maximum lengths of the paths actually generated by Algorithm S2S. Moreover, the average path length is almost equal to $n/3$.

Figs. 10 and 11 show the average execution times of our algorithm in 0-MC_n and 1-MC_n, respectively. From these figures, we can observe that there is not a significant difference between 0-MC_n and 1-MC_n and the average execution times are approximately converging to $O(n^2)$.

VI. CONCLUSION AND FUTURE WORKS

In this paper, we proposed a polynomial-order time algorithm for the set-to-set disjoint paths problem in n-Möbius cubes. Its time complexity is $O(n^6)$ and the maximum path length is $2n-2$. These results show that our algorithm outperforms the algorithm proposed by Kaneko *et al.* [57] regarding both the time complexity and the path length. We also conducted a computer experiment and showed that the maximum path length is about $n/2$, the average path length is about $n/3$, and the average execution time is $O(n^2)$. From these experimental results, the maximum lengths and the average lengths of generated paths are almost equal to the diameter and the average distance of the Möbius cube, respectively, and the average execution time complexity is $O(n^2)$, which is necessary to find n shortest paths in an n-Möbius cube. Hence, we can conclude that our algorithm is not only theoretically correct but also practically useful on average in massively parallel systems.

Future works include theoretical analysis of the maximum path length and the average performance of the algorithm. Also, improvement of the algorithm to generate shorter paths in smaller execution time is also interesting for us.

ACKNOWLEDGMENT

The authors would like to express special thanks to Dr.Bipin Indurkhya for his proofreading the draft of this paper.

APPENDIX
ALGORITHM SP

We give a pseudo code of Algorithm SP in Fig. 12. Note that $\boldsymbol{e}_i = (e_1, e_2, \ldots, e_n)$ $(1 \leq i \leq n)$ and $\boldsymbol{E}_i = (E_1, E_2, \ldots, E_n)$ $(1 \leq i \leq n)$ where $e_j = 0$ if $j \neq i$ and $e_j = 1$ if $j = i$, and $E_j = 0$ if $j < i$ and $E_j = 1$ if $j \geq i$. For a source vertex $\boldsymbol{s}$ and a target vertex $\boldsymbol{t}$ in an n-Möbius cube, we call the function `SP` by `SP` $(\boldsymbol{s}, \boldsymbol{t})$. Then, `SP` will return one of the shortest paths from $\boldsymbol{s}$ to $\boldsymbol{t}$.

REFERENCES

[1] P. A. J. Hilbers, M. R. Koopman, and J. L. A. van de Snepscheut, "The twisted cube," in *PARLE Parallel Architectures and Languages Europe*, vol. 1. London, U.K.: Springer-Verlag, 1987, pp. 152–159. [Online]. Available: http://dl.acm.org/citation.cfm?id=25489.25499

[2] A. El-Amawy and S. Latifi, "Properties and performance of folded hypercubes," *IEEE Trans. Parallel Distrib. Syst.*, vol. 2, no. 1, pp. 31–42, Jan. 1991.

[3] K. Efe, "The crossed cube architecture for parallel computation," *IEEE Trans. Parallel Distrib. Syst.*, vol. 3, no. 5, pp. 513–524, Sep. 1992.

[4] W. Zhou, J. Fan, X. Jia, and S. Zhang, "The spined cube: A new hypercube variant with smaller diameter," *Inf. Process. Lett.*, vol. 111, no. 12, pp. 561–567, Jun. 2011.

[5] P. Kulasinghe and S. Bettayeb, "Embedding binary trees into crossed cubes," *IEEE Trans. Comput.*, vol. 44, no. 7, pp. 923–929, Jul. 1995.

[6] C.-P. Chang, J.-N. Wang, and L.-H. Hsu, "Topological properties of twisted cube," *Inf. Sci.*, vol. 113, nos. 1–2, pp. 147–167, Jan. 1999.

[7] J.-S. Fu, "Fault-free cycles in folded hypercubes with more faulty elements," *Inf. Process. Lett.*, vol. 108, no. 5, pp. 261–263, Nov. 2008.

[8] P.-L. Lai and H.-C. Hsu, "Constructing the nearly shortest path in crossed cubes," *Inf. Sci.*, vol. 179, no. 14, pp. 2487–2493, Jun. 2009.

[9] J.-M. Chang, J.-D. Wang, J.-S. Yang, and K.-J. Pai, "A comment on 'independent spanning trees in crossed cubes,'" *Inf. Process. Lett.*, vol. 114, no. 12, pp. 734–739, 2014.

[10] G.-L. Cheng, Q. Zhu, and X.-K. Wang, "On the reliability and fault tolerance of spined cubes," in *Proc. Int. Conf. Wavelet Anal. Pattern Recognit.*, Jul. 2012, pp. 313–316.

[11] J.-M. Chang, T.-J. Yang, and J.-S. Yang, "A parallel algorithm for constructing independent spanning trees in twisted cubes," *Discrete Appl. Math.*, vol. 219, pp. 74–82, Mar. 2017. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0166218X16305704

[12] H.-C. Chen, T.-L. Kung, and L.-H. Hsu, "An augmented pancyclicity problem of crossed cubes," *Comput. J.*, vol. 61, no. 1, pp. 54–62, Jan. 2018.

[13] C.-N. Kuo and Y.-H. Cheng, "Cycles in folded hypercubes with two adjacent faulty vertices," *Theor. Comput. Sci.*, vol. 795, pp. 115–118, Nov. 2019. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0304397519303913

[14] C. L. Seitz, "The cosmic cube," *Commun. ACM*, vol. 28, no. 1, pp. 22–33, Jan. 1985.

[15] P. Cull and S. M. Larson, "The Möbius cubes," *IEEE Trans. Comput.*, vol. 44, no. 5, pp. 647–659, May 1995.

[16] J. Fan, "Hamilton-connectivity and cycle-embedding of the Möbius cubes," *Inf. Process. Lett.*, vol. 82, no. 2, pp. 113–117, Apr. 2002.

[17] S.-Y. Hsieh and C.-H. Chen, "Pancyclicity on Möbius cubes with maximal edge faults," *Parallel Comput.*, vol. 30, no. 3, pp. 407–421, Mar. 2004.

[18] D. Kocík, Y. Hirai, and K. Kaneko, "Node-to-set disjoint paths problem in a Möbius cube," *IEICE Trans. Inf. Syst.*, vol. E99-D, no. 3, pp. 708–713, Mar. 2016.

[19] C.-H. Tsai, "Embedding of meshes in Möbius cubes," *Theor. Comput. Sci.*, vol. 401, nos. 1–3, pp. 181–190, Jul. 2008.

[20] J.-M. Xu, M. Ma, and M. Lü, "Paths in Möbius cubes and crossed cubes," *Inf. Process. Lett.*, vol. 97, no. 3, pp. 94–97, Feb. 2006.

[21] X. Yang, G. M. Megson, and D. J. Evans, "Pancyclicity of Möbius cubes with faulty nodes," *Microprocessors Microsyst.*, vol. 30, no. 3, pp. 165–172, May 2006.

[22] J. Duato, S. Yalamanchili, and L. Ni, *Interconnection Networks: An Engineering Approach*. San Francisco, CA, USA: Morgan Kaufmann, 2002.

[23] A. Bossard, "A set-to-set disjoint paths routing algorithm in hyper-star graphs," *Int. J. Comput. Their Appl.*, vol. 21, no. 1, pp. 76–82, Mar. 2014.

[24] A. Bossard and K. Kaneko, "The set-to-set disjoint-path problem in perfect hierarchical hypercubes," *Comput. J.*, vol. 55, no. 6, pp. 769–775, Jun. 2012.

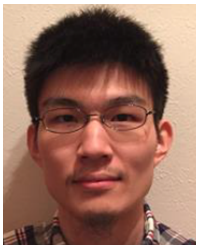
[25] X.-B. Chen, "Many-to-many disjoint paths in faulty hypercubes," *Inf. Sci.*, vol. 179, no. 18, pp. 3110–3115, Aug. 2009.

[26] Q.-P. Gu and S. Peng, "Set-to-set fault tolerant routing in star graphs," *IEICE Trans. Inf. Syst.*, vol. E79-D, no. 4, pp. 282–289, Apr. 1996.

[27] Q.-P. Gu and S. Peng, "Node-to-set and set-to-set cluster fault tolerant routing in hypercubes," *Parallel Comput.*, vol. 24, no. 8, pp. 1245–1261, Aug. 1998.

[28] X.-J. Li, B. Liu, M. Ma, and J.-M. Xu, "Many-to-many disjoint paths in hypercubes with faulty vertices," *Discrete Appl. Math.*, vol. 217, pp. 229–242, Jan. 2017. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0166218X1630405X

[29] J. Li, C. Melekian, S. Zuo, and E. Cheng, "Unpaired many-to-many disjoint path covers on bipartite k-ary n-cube networks with faulty elements," *Int. J. Found. Comput. Sci.*, vol. 31, no. 3, pp. 371–383, Apr. 2020, doi: 10.1142/S0129054120500148.
[30] J. Arai and Y. Li, "Disjoint-path routing on hierarchical dual-nets," *Int. J. Netw. Comput.*, vol. 4, no. 2, pp. 260–278, Jul. 2014.
[31] M. Dietzfelbinger, S. Madhavapeddy, and I. H. Sudborough, "Three disjoint path paradigms in star networks," in *Proc. IEEE Symp. Parallel Distrib. Process.*, Dec. 1991, pp. 400–406.
[32] J.-S. Fu, G.-H. Chen, and D.-R. Duh, "Node-disjoint paths and related problems on hierarchical cubic networks," *Networks*, vol. 40, no. 3, pp. 142–154, Oct. 2002.
[33] Y. Hamada, F. Bao, A. Mei, and Y. Igarashi, "Nonadaptive fault-tolerant file transmission in rotator graphs," *IEICE Trans. Fundam.*, vol. E79-A, no. 4, pp. 477–482, Apr. 1996.
[34] D. Kocík and K. Kaneko, "Node-to-node disjoint paths problem in a Möbius cube," *IEICE Trans. Inf. Syst.*, vol. E100-D, no. 8, pp. 1837–1843, Aug. 2017.
[35] H. Liu, "The construction of disjoint paths in folded hypercube," *J. Syst. Sci. Inf.*, vol. 8, no. 2, pp. 97–102, Jun. 2010.
[36] T.-C. Lin and D.-R. Duh, "Constructing vertex–disjoint paths in (n, k)-star graphs," *Inf. Sci.*, vol. 178, no. 3, pp. 788–801, Feb. 2008.
[37] S. Madhavapeddy and I. H. Sudborough, "A topological property of hypercubes: Node disjoint paths," in *Proc. 2nd IEEE Symp. Parallel Distrib. Process.*, Dec. 1990, pp. 532–539.
[38] Y.-K. Shih and S.-S. Kao, "One-to-one disjoint path covers on k-ary n-cubes," *Theor. Comput. Sci.*, vol. 412, no. 35, pp. 4513–4530, Aug. 2011.
[39] O. Alsaleh, B. Bose, and B. Hamdaoui, "One-to-many node-disjoint paths routing in dense Gaussian networks," *Comput. J.*, vol. 58, no. 2, pp. 173–187, Feb. 2015.
[40] D. Choi and I. Chung, "A nearly optimal one-to-many routing algorithm on k-ary n-cube networks," *Korean Inst. Smart Media*, vol. 7, no. 2, pp. 9–14, May 2018.
[41] S. Fujita, "Polynomial time algorithm for constructing vertex-disjoint paths in transposition graphs," *Networks*, vol. 56, no. 2, pp. 149–157, Aug. 2010.
[42] Y.-S. Huang, S.-L. Peng, and R.-Y. Wu, "An improved algorithm for the node-to-set disjoint paths problem on bi-rotator graphs," *J. Interconnection Netw.*, vol. 16, nos. 3–4, pp. 1–18, 2016.
[43] Q.-P. Gu and S. Peng, "Node-to-set disjoint paths problem in star graphs," *Inf. Process. Lett.*, vol. 62, no. 4, pp. 201–207, May 1997.
[44] Y. Li, S. Peng, and W. Chu, "Disjoint-paths and fault-tolerant routing on recursive dual-net," *Int. J. Found. Comput. Sci.*, vol. 22, no. 5, pp. 1001–1018, Aug. 2011.
[45] S. Ling and W. Chen, "Node-to-set disjoint paths in biswapped networks," *Comput. J.*, vol. 57, no. 7, pp. 953–967, Jul. 2014.
[46] C.-N. Lai, G.-H. Chen, and D.-R. Duh, "Constructing one-to-many disjoint paths in folded hypercubes," *IEEE Trans. Comput.*, vol. 51, no. 1, pp. 33–45, Jan. 2002.
[47] L. Lipták, E. Cheng, J.-S. Kim, and S. W. Kim, "One-to-many node-disjoint paths of hyper-star networks," *Discrete Appl. Math.*, vol. 160, nos. 13–14, pp. 2006–2014, Sep. 2012.
[48] X. Wang, J. Fan, S. Zhang, and J. Yu, "Node-to-set disjoint paths problem in cross-cube," *J. Supercomput.*, vol. 78, no. 1, pp. 1356–1380, Jan. 2022.
[49] Y. Xiang and I. A. Stewart, "One-to-many node-disjoint paths in (n, k)-star graphs," *Discrete Appl. Math.*, vol. 158, no. 1, pp. 62–70, Jan. 2010.
[50] A. Bossard and K. Kaneko, "k-pairwise disjoint paths routing in perfect hierarchical hypercubes," *J. Supercomput.*, vol. 67, no. 2, pp. 485–495, Feb. 2014.
[51] Q.-P. Gu and S. Peng, "K-pairwise cluster fault tolerant routing in hypercubes," *IEEE Trans. Comput.*, vol. 46, no. 9, pp. 1042–1049, Sep. 1997.
[52] Q.-P. Gu and S. Peng, "An efficient algorithm for k-pairwise disjoint paths in star graphs," *Inf. Process. Lett.*, vol. 67, no. 6, pp. 283–287, Sep. 1998.
[53] Q.-P. Gu and S. Peng, "An efficient algorithm for the k-pairwise disjoint paths problem in hypercubes," *J. Parallel Distrib. Comput.*, vol. 60, no. 6, pp. 764–774, Jun. 2000.
[54] K. Kaneko, S. V. Nguyen, and H. T. T. Binh, "Pairwise disjoint paths routing in tori," *IEEE Access*, vol. 8, pp. 192206–192217, 2020.
[55] K. Menger, "Zur allgemeinen Kurventhoerie," *Fundamenta Mathematicae*, vol. 10, no. 1, pp. 96–115, 1927.
[56] K. Kaneko, "An algorithm for set-to-set disjoint paths problem in a Möbius cube," in *Proc. Int. Conf. Parallel Distrib. Process. Techn. Appl.*, Jul. 2017, pp. 39–45.
[57] K. Kaneko, A. Bossard, and F. C. Harris, Jr., "Set-to-set disjoint path routing in bijective connection graphs," *IEEE Access*, vol. 10, pp. 72731–72742, 2022.



HIROYUKI ICHIDA received the B.E. and M.E. degrees from the Tokyo University of Agriculture and Technology, Japan, in 2019 and 2021, respectively, where he is currently pursuing the Ph.D. degree with the Department of Electronic and Information Engineering, Graduate School of Engineering.

His research interests include dependable systems and graph theory.

KEIICHI KANEKO (Member, IEEE) received the B.E., M.E., and Ph.D. degrees in engineering from the University of Tokyo, in 1985, 1987, and 1994, respectively.

He is currently a Professor with the Tokyo University of Agriculture and Technology, Japan. His research interests include functional programming, parallel and distributed computation, partial evaluation, and dependable systems. He is a member of ACM, IEEE CS, IEICE, IPSJ, and JSSST.