Node-to-Node Disjoint Paths Problem in Möbius Cubes

David KOČÍK†, Nonmember and Keiichi KANEKO††, Member

SUMMARY The Möbius cube is a variant of the hypercube. Its advantage is that it can connect the same number of nodes as a hypercube but with almost half the diameter of the hypercube. We propose an algorithm to solve the node-to-node disjoint paths problem in n-Möbius cubes in polynomial-order time of n. We provide a proof of correctness of the algorithm and estimate that the time complexity is O(n²) and the maximum path length is 3n − 5.

key words: container problem, hypercube, multicomputer, interconnection network, parallel processing, dependable computing, performance evaluation

1. Introduction

The Möbius cube [5] is a variant of the hypercube [21]. Its advantage is that it can connect the same number of nodes as a hypercube but with almost half the diameter of the hypercube. Therefore, it is a promising topology [6], [12], [16], [23]–[25]. One major issue with the Möbius cubes is the node-to-node disjoint-paths problem: for two distinct nodes in a k-connected graph, find k paths between them that are disjoint except for the nodes. This, together with the node-to-set disjoint paths problem [8], [13], [14] and the set-to-set disjoint paths problem [3], [7], [9] are all considered to be major issues in parallel and distributed computation [2], [10], [18], [22].

In general, for an arbitrary graph G(V,E), the node-to-node disjoint paths can be obtained in polynomial-order time of |V| by using the maximum flow algorithm. However, the complexity of the algorithm is too large for an n-dimensional Möbius cube, or an Mn in short, because the number of nodes in it is equal to 2n. Kočík et al. have proposed an algorithm that solves this problem in polynomial time of n [15]. That is, the time complexity of their algorithm is O(n²), and the maximum length of the generated paths is O(n²). They have also proposed an algorithm to solve the node-to-set disjoint paths problem [16], which is applicable to the node-to-node disjoint paths problem. The time complexity of the algorithm is O(n²) and the maximum path length is O(n). In this paper, we give an algorithm N2N that solves the same problem in the time complexity O(n²) while the maximum length of the paths is O(n).

For the hypercube and its variants, the node-to-node disjoint paths problem has been extensively studied before. For an n-dimensional hypercube of diameter n, Saad and Shultz [20] have proposed an algorithm of O(n²) time that generates n disjoint paths whose lengths are at most n + 1. For an n-dimensional crossed cube of diameter [(n + 1)/2], Kulasinghe [17] has proposed an algorithm of O(n²) time that generates n disjoint paths whose lengths are at most 3n − 5. For an n-dimensional twisted cube of diameter [(n + 1)/2], Chang et al. [4] have proposed an algorithm of O(n²) time that generates n disjoint paths whose lengths are at most (n/2) + 1. For an n-dimensional folded hypercube of diameter [n/2], Liu [19] has proposed an algorithm of O(n²) time that generates (n + 1) disjoint paths whose lengths are at most [n/2] + 2. The maximum path length by our research is 3n − 5, which is equal to the result for an n-dimensional crossed cube. Although several such studies have attained optimal or quasi optimal solutions for the maximum path lengths compared to the diameters, there is a significant gap between the maximum path length by our research and the diameter [(n + 2)/2] or [(n + 1)/2] of an n-dimensional Möbius cube.

The rest of this paper is organized as follows. In Sect. 2, we define the n-dimensional Möbius cube and present three lemmas related to its properties. Next, in Sect. 3, we explain our algorithm N2N in detail. Then, in Sect. 4, we present the correctness proof of N2N and discuss its theoretical complexities. Finally, we give a conclusion and future works in Sect. 5.

2. Preliminaries

We define below the Möbius cube and present three lemmas related to its properties.

Definition 1: An n-dimensional Möbius cube Mn has 2n nodes. Each node has a unique n-bit address. For two nodes a = (a1, a2, ..., an) and b, they are connected if and only if one of the following conditions is satisfied:

\[ b = \begin{cases} (a_1, a_2, ..., a_{i-1}, \bar{a}_i, a_{i+1}, ..., a_n) & \text{if } a_{i-1} = 0, \\ (a_1, a_2, ..., a_{i-1}, \bar{a}_i, \bar{a}_{i+1}, ..., \bar{a}_n) & \text{if } a_{i-1} = 1. \end{cases} \]

where we can assume that a0 = 0 or a0 = 1. For the former case, we call it an n-dimensional 0-Möbius cube, 0-Mn, while for the latter an n-dimensional 1-Möbius cube, 1-Mn.
If two nodes $a$ and $b$ are connected by one of the conditions in Definition 1, we say that $a$ and $b$ are connected by an edge of $i$-th dimension, and $b$ is denoted by $a^i$ or $a$ is denoted by $b^{(i)}$.

Figure 1 depicts an example each of $0$-$M_4$ and $1$-$M_4$. Note that a $0$-$M_4$ and a $1$-$M_4$ provide different topologies. For example, the average distance for a $0$-$M_4$ is equal to $1.81$ while that for a $1$-$M_4$ is equal to $1.75$ [16]. An $M_n$ consists of two disjoint subgraphs $M^0$ and $M^1$ where an $M^i$ is a subgraph derived from the set of nodes $\{x = (x_1, x_2, \ldots, x_n) \mid x_1 = i\}$. Note that an $M^0$ and an $M^1$ are isomorphic to a $0$-$M_{n-1}$ and a $1$-$M_{n-1}$, respectively.

Unfortunately, the Möbius cubes $M_n$ ($n \geq 4$) are neither node symmetric nor edge symmetric. Hence, their practical use has limitations. Cull and Larson [5] have proposed simulation of a hypercube program on a Möbius cube topology. In their approach, one routing step in the hypercube can be implemented by at most two steps in the Möbius cube.

Table 1 shows a comparison of the characteristics of a $0$-$M_n$ and a $1$-$M_n$ with an $n$-dimensional twisted hypercube, $T_n$ [11]. With respect to the diameter, a $T_n$ is a slightly better than a $0$-$M_n$. However, a $T_n$ is much inferior to a $0$-$M_n$ and a $1$-$M_n$ with respect to the average distance.

There is a shortest-path routing algorithm for an $M_n$ and it takes $O(n)$ time [5]. In the rest of this paper, we refer this algorithm spr.

We assume that each node address is stored in a machine word, and generating an edge by obtaining $a^{(i)}$ for any node $a$ requires $O(1)$ time. Then, in an $M_n$, spr takes $O(n)$ execution time to generate a shortest path whose length is at most $\lceil (n + 2)/2 \rceil$ [5].

**Lemma 1:** For any node $a$ in a $0$-$M_n$ where $n \geq 4$, we can generate $(n - 1)$ paths $Q_i$: $a^{(i)} \rightarrow a^{(i+1)}$ ($2 \leq i \leq n$) that are disjoint other than $a^{(1)}$. The time complexity to generate these $(n - 1)$ paths is $O(n)$. The length of any of the paths is at most $4$.

(Proof) First, let us consider the $(n - 2)$ paths $Q_i$ ($3 \leq i \leq n$). Then, path generation of $Q_i$ is divided into the following two cases, Case 1 and Case 2, depending on the value of $a_{i-1}$.

**Case 1** ($a_{i-1} = 0$): Generate a path $Q_i$: $a^{(i)} = (a_1, a_2, \ldots, a_{i-1}, \bar{a}_i, a_{i+1}, \ldots, a_n)$ $\rightarrow a^{(i+1)} = (\bar{a}_1, a_2, \ldots, a_{i-1}, \bar{a}_i, a_{i+1}, \ldots, a_n) = a^{(1)} \rightarrow a^{(1)}$.

**Case 2** ($a_{i-1} = 1$): Generate a path $Q_i$: $a^{(i)} = (a_1, a_2, \ldots, a_{i-1}, a_i, a_{i+1}, \ldots, a_n)$ $\rightarrow a^{(i+1)} = (\bar{a}_1, a_2, \ldots, a_{i-1}, a_i, a_{i+1}, \ldots, a_n) = a^{(1)} \rightarrow a^{(1)}$.

Next, let us consider the remaining path $Q_2$. Then, path generation of $Q_2$ is divided into the following six cases, Cases 3 to 8, depending on the values of $a_1, a_2,$ and $a_3$.

**Case 3** ($a_1 = a_2 = 0$): Generate a path $Q_2$: $a^{(2)} = (a_1, \bar{a}_2, a_3, \ldots, a_n)$ $\rightarrow a^{(2,1)} = (\bar{a}_1, \bar{a}_2, a_3, \ldots, a_n) = a^{(2)} \rightarrow a^{(1)}$.

**Case 4** ($a_1 = 0, a_2 = 1, a_3 = 0$): Generate a path $Q_2$: $a^{(2)} = (a_1, a_2, a_3, \ldots, a_n)$ $\rightarrow a^{(2,1)} = (\bar{a}_1, a_2, a_3, \ldots, a_n) = a^{(2)} \rightarrow a^{(1)}$.

**Case 5** ($a_1 = 0, a_2 = a_3 = 1$): Generate a path $Q_2$: $a^{(2)} = (a_1, \bar{a}_2, a_3, \ldots, a_n)$ $\rightarrow a^{(2,1)} = (\bar{a}_1, \bar{a}_2, a_3, \ldots, a_n) = a^{(2)} \rightarrow a^{(1)}$.

**Case 6** ($a_1 = 1, a_2 = 0$): Generate a path $Q_2$: $a^{(2)} = (a_1, \bar{a}_2, a_3, \ldots, a_n)$ $\rightarrow a^{(2,1)} = (\bar{a}_1, \bar{a}_2, a_3, \ldots, a_n) = a^{(2)} \rightarrow a^{(1)}$.

**Case 7** ($a_1 = a_2 = 1, a_3 = 0$): Generate a path $Q_2$: $a^{(2)} = (a_1, a_2, a_3, \ldots, a_n)$ $\rightarrow a^{(2,1)} = (\bar{a}_1, a_2, a_3, \ldots, a_n) = a^{(2)} \rightarrow a^{(1)}$.

**Case 8** ($a_1 = a_2 = a_3 = 1$): Generate a path $Q_2$: $a^{(2)} = (a_1, a_2, a_3, \ldots, a_n)$ $\rightarrow a^{(2,1)} = (\bar{a}_1, a_2, a_3, \ldots, a_n) = a^{(2)} \rightarrow a^{(1)}$.

The $(n - 2)$ paths generated in Case 1 or Case 2 are $Q_i$: $a^{(i)} \rightarrow a^{(i+1)}$ ($3 \leq i \leq n$). Each internal node $a^{(1)}$ has its unique pattern with preceding $i$ bits: $(\bar{a}_1, a_2, \ldots, a_{i-1}, a_i, \ldots)$. Hence, the paths $Q_i$ ($3 \leq i \leq n$) are disjoint other than $a^{(1)}$. The path generated in Case 3, 4, 5, 6, 7, 8 is $Q_{n-2}$.
For any node $a$ in a 1-$M_n$, where $n \geq 4$, we can generate $(n-1)$ paths $Q_i$: $a^{(0)} \leadsto a^{(1)} (2 \leq i \leq n)$ that are disjoint other than $a^{(1)}$. The time complexity to generate these paths is $O(n^2)$. The length of any of the paths is at most 4.

(Proof) First, let us consider the $(n-2)$ paths $Q_i$ ($2 \leq i \leq n-1$). Then, path generation of $Q_i$ is divided into the following six cases, Cases 1 to 6, depending on the values of $a_{i-1}$, $a_i$, and $a_{i+1}$.

**Case 1** ($a_{i-1} = a_i = a_{i+1} = 0$) Generate a path $Q_i$: $a^{(0)} = (a_{1}, a_{2}, \ldots, a_{i}, \tilde{a}_{i}, a_{i+1}, \ldots, a_n) \rightarrow a^{(1)} = (\tilde{a}_{1}, \tilde{a}_{2}, \ldots, \tilde{a}_{i-1}, a_{i+1}, a_{i+2}, \ldots, a_n) = a^{(1,1)} \rightarrow a^{(1)}$.

**Case 2** ($a_{i-1} = a_i = 0, a_{i+1} = 1$) Generate a path $Q_i$: $a^{(0)} = (a_{1}, a_{2}, \ldots, a_{i}, \tilde{a}_{i}, a_{i+1}, \ldots, a_n) \rightarrow a^{(1)} = (\tilde{a}_{1}, a_{2}, \ldots, \tilde{a}_{i-1}, a_i, a_{i+1}, a_{i+2}, \ldots, a_n) = a^{(1,2)} \rightarrow a^{(1)}$.

**Case 3** ($a_{i-1} = 0, a_{i+1} = 1$) Generate a path $Q_i$: $a^{(0)} = (a_{1}, a_{2}, \ldots, \tilde{a}_{i-1}, a_i, a_{i+1}, \ldots, a_n) \rightarrow a^{(1)} = (\tilde{a}_{1}, \tilde{a}_{2}, \ldots, \tilde{a}_{i-1}, a_i, a_{i+1}, a_{i+2}, \ldots, a_n) = a^{(1,3)} \rightarrow a^{(1)}$.

**Case 4** ($a_{i-1} = 1, a_i = a_{i+1} = 0$) Generate a path $Q_i$: $a^{(0)} = (a_{1}, a_{2}, \ldots, a_{i}, \tilde{a}_{i}, a_{i+1}, \ldots, a_n) \rightarrow a^{(1)} = (\tilde{a}_{1}, a_{2}, \ldots, \tilde{a}_{i-1}, a_i, a_{i+1}, a_{i+2}, \ldots, a_n) = a^{(1,4)} \rightarrow a^{(1)}$.

**Case 5** ($a_{i-1} = 1, a_i = 0, a_{i+1} = 1$) Generate a path $Q_i$: $a^{(0)} = (a_{1}, a_{2}, \ldots, \tilde{a}_{i-1}, a_i, a_{i+1}, \ldots, a_n) \rightarrow a^{(1)} = (\tilde{a}_{1}, a_{2}, \ldots, \tilde{a}_{i-1}, a_i, a_{i+1}, a_{i+2}, \ldots, a_n) = a^{(1,5)} \rightarrow a^{(1)}$.

**Case 6** ($a_{i-1} = 1, a_i = 1$) Generate a path $Q_i$: $a^{(0)} = (a_{1}, a_{2}, \ldots, a_{i-1}, \tilde{a}_{i}, a_{i+1}, \ldots, a_n) \rightarrow a^{(1)} = (\tilde{a}_{1}, a_{2}, \ldots, \tilde{a}_{i-1}, a_i, a_{i+1}, a_{i+2}, \ldots, a_n) = a^{(1,6)} \rightarrow a^{(1)}$.

Next, let us consider the remaining path $Q_n$. We can generate $Q_n$ as follows.

**Case 7** Generate a path $Q_n$: $a^{(0)} = (a_1, a_2, \ldots, a_{n-1}, \tilde{a}_n) \rightarrow a^{(1,1)} = (\tilde{a}_1, a_2, \ldots, a_{n-1}, a_n) = a^{(1,0)} \rightarrow a^{(1)}$.

The $(n-1)$ paths are $Q_i$: $a^{(0)} \leadsto a^{(1,1)} \leadsto a^{(1,2)} \leadsto a^{(1)} (2 \leq i \leq n)$. Since each subpath $a^{(1,1)} \leadsto a^{(1,2)}$ is included in the $a_{i-1}M_n$, and the internal nodes of $Q_i$ are all in the form of $(\tilde{a}_1, a_2, \ldots, a_{i-1}, a_i, \ldots)$. $Q_i$'s $(2 \leq i \leq n)$ are disjoint other than $a^{(1)}$ with each other.

It takes $O(1)$ time to generate one of the paths $Q_i (2 \leq i \leq n)$. In total, the $(n-1)$ paths are generated in $O(n)$ time. The length of any of the paths is at most 4. See Fig. 3.

**Lemma 3:** For any node $a$ in an $M_n$, where $n \geq 4$, we can generate $(n-1)$ paths $Q_i$: $a^{(0)} \leadsto a^{(1)} (2 \leq i \leq n)$ that are disjoint other than $a^{(1)}$. The time complexity to generate these paths is $O(n^2)$. The length of any of the paths is at most 4. See Fig. 3.

### 3. Algorithm N2N

In this section, we give an algorithm N2N that solves the node-to-node disjoint paths problem for the Möbius cube. In other words, for a source node $s$ and a destination node $d$ in an $M_n$, N2N finds $n$ paths from $s$ to $d$ that are disjoint other than $s$. For an $M_n$, we can generate $n$ disjoint paths of lengths at most 4 between any pair of nodes by enumeration. Hence, we assume that $n \geq 4$ in the rest of this section.

#### 3.1 Procedure 1

First, we assume that the source node and the destination node are included in the same $M^j (j \in [0, 1])$, that is, $s, d \in M^j$. Then, we generate $n$ paths from $s$ to $d$ that are disjoint other than $s$ and $d$ by the following steps.

**Step 1** Apply the algorithm recursively in the $M^j$ to generate $(n-1)$ paths from $s$ to $d$ that are disjoint other than $s$ and $d$.

**Step 2** Choose edges $s \rightarrow s^{(1)}$ and $d \rightarrow d^{(1)}$.

**Step 3** Generate a path $s^{(1)} \leadsto d^{(1)}$ in the $M^j$ by using the algorithm spr. See Fig. 4.

#### 3.2 Procedure 2

Next, we assume that the source node is included in an $M^j$...
3.3 Procedure 3

Now, we assume that the source node is included in an $M^i$ ($d \in M^i$), and $s = d^{(1)}$. Then, we generate $n$ paths from $s$ to $d$ that are disjoint other than $s$ and $d$ by the following steps.

Step 1 Choose $n$ edges $s \to s^{(i)}$ ($1 \leq i \leq n$).

Step 2 As shown in the proof of Lemma 3, generate $(n-1)$ paths $Q_i: s^{(i)} \leadsto d$ ($2 \leq i \leq n$). Then, we can have $n$ paths $R_i$ ($1 \leq i \leq n$): 

\[ R_i : \begin{cases} 
    s \to d & \text{if } i = 1, \\
    s \to s^{(i)} \leadsto d^{(i)} \to d & \text{if } 2 \leq i \leq n.
\end{cases} \]

See Fig. 5.

3.4 Procedure 4

Finally, we assume that the source node is included in an $M^i$ ($s \in M^i$), the destination node is included in an $M^j$ ($d \in M^j$), and the distance between $s$ and $d^{(1)}$ is more than 1. Then, we generate $n$ paths from $s$ to $d$ that are disjoint other than $s$ and $d$ by the following steps.

Step 1 Apply the algorithm recursively in the $M^j$ to generate $(n-1)$ paths $P_i: s \leadsto d^{(i)}$ ($2 \leq i \leq n$) that are disjoint other than $s$ and $d^{(1)}$.

Step 2 As shown in the proof of Lemma 3, generate $(n-1)$ paths $Q_i : d^{(1)} \leadsto d$ ($2 \leq i \leq n$).

Step 3 Assuming that $s = d^{(1)}$, delete edges $d^{(1)} \to d^{(i)}$ ($2 \leq i \neq h \leq n$) and choose an edge $d^{(1)} \to d$. Then, we can have $n$ paths $R_i$ ($1 \leq i \leq n$): 

\[ R_i : \begin{cases} 
    s \to s^{(1)} \leadsto d^{(i)} \to d & \text{if } i = 1, \\
    s \to d^{(1)} \leadsto d^{(1,i-1)} \leadsto d^{(i)} & \text{if } 2 \leq i \neq h \leq n, \\
    s \to d^{(1)} \to d & \text{if } i = h.
\end{cases} \]

See Fig. 6.

IEICE TRANS. INF. & SYST., VOL.E100–D, NO.8 AUGUST 2017

Fig. 4 After Step 3 in Procedure 1.

Fig. 5 After Step 2 in Procedure 2.

Fig. 6 After Step 3 in Procedure 3.

Fig. 7 After Step 1 in Procedure 4.

Fig. 8 After Step 2 in Procedure 4.

Fig. 9 After Step 5 in Procedure 4.
Step 6 Delete the subpath \( d^{1,h} \rightarrow d^{1,h,1} \rightarrow u \) and the \((n - 2)\) edges \( d^{1,2} \rightarrow d^{1(1)} \) included in \( P_i \) \((2 \leq i \neq h \leq n)\).

Consequently, we can have \( n \) paths \( R_i \) \((1 \leq i \leq n)\):

\[
R_i : \begin{cases} 
    s \rightarrow s^{(1)} \rightarrow u \rightarrow d^{(h)} \rightarrow d & \text{if } i = 1, \\
    s \rightarrow d^{(1,2)} \rightarrow d^{(1,2,1)} \rightarrow d^{(1,0)} \rightarrow d & \text{if } 2 \leq i \neq h \leq n, \\
    s \rightarrow d^{(1,h)} \rightarrow d^{(1)} \rightarrow d & \text{if } i = h.
\end{cases}
\]

See Fig. 10.

4. Proof of Correctness and Estimation of Complexities

We provide a proof of correctness of Algorithm N2N. Also, we estimate the time complexity \( T(n) \) and the maximum path length \( L(n) \) of the algorithm for an \( n \)-dimensional Möbius cube. The proof is based on induction on \( n \).

**Lemma 4:** For two nodes \( s \) and \( d \) in an \( M_n \), Procedure 1 generates \( n \) paths from \( s \) to \( d \) that are disjoint other than \( s \) and \( d \). The time complexity of Procedure 1 is \( T(n - 1) + O(n) \) and the maximum length of the paths generated is \( \max[L(n - 1), [n/2] + 3] \).

(Proof) By hypothesis of induction, the paths generated in Step 1 are disjoint other than \( s \) and \( d \). The path generated in Steps 2 and 3 is outside of \( M^j \) other than \( s \) and \( d \). Hence, the path does not have any common node with the paths generated in Step 1 other than \( s \) \( d \). That is, it is disjoint other than \( s \) \( d \) with other paths generated in Step 1.

By hypothesis of induction, Step 1 takes \( T(n - 1) \) time to generate \((n - 1)\) paths and the maximum length of them is \( L(n - 1) \). The path generated in Steps 2 and 3 consists of two edges and a subpath by spr. Therefore, Steps 2 and 3 take \( O(n) \) time to generate the path whose length is at most \( 2 + \lceil (n + 1)/2 \rceil = [n/2] + 3 \).

Consequently, the time complexity of Procedure 1 is \( T(n - 1) + O(n) \) and the maximum path length is \( \max[L(n - 1), [n/2] + 3] \).

**Lemma 5:** For two nodes \( s \) and \( d \) in an \( M_n \), Procedure 2 generates \( n \) paths from \( s \) to \( d \) that are disjoint other than \( s \) and \( d \). The time complexity of Procedure 2 is \( O(n) \) and the maximum length of the paths generated is \( 5 \).

(Proof) The \( n \) edges \( s \rightarrow s^{(1)} \) \((1 \leq i \leq n)\) chosen in Step 1 are disjoint other than \( s \). From Lemma 3, the paths \( Q_i \) \((2 \leq i \leq n)\) generated in Step 2 are disjoint other than \( d \) with each other. The path \( R_1 : s \rightarrow d \) does not have any node other than \( s \) and \( d \). Therefore, the \( n \) paths \( R_i \) \((1 \leq i \leq n)\) are disjoint other than \( s \) and \( d \) with each other.

The time complexity for choosing the \( n \) edges in Step 1 is \( O(n) \). From Lemma 3, Step 2 takes \( O(n) \) time to generate the \((n - 1)\) paths \( Q_i \) \((2 \leq i \leq n)\) of lengths at most \( 4 \). Consequently, in total, Procedure 2 takes \( O(n) \) time to generate the \( n \) paths \( R_i \) \((1 \leq i \leq n)\) whose lengths are at most \( 1 + 4 = 5 \).

**Lemma 6:** For two nodes \( s \) and \( d \) in an \( M_n \), Procedure 3 generates \( n \) paths from \( s \) to \( d \) that are disjoint other than \( s \) and \( d \). The time complexity of Procedure 3 is \( T(n - 1) + O(n) \) and the maximum length of the paths generated is \( L(n - 1) + 3 \).

(Proof) By hypothesis of induction, the \((n - 1)\) paths \( P_i \) \((2 \leq i \leq n)\) generated in Step 1 are disjoint other than \( s \) and \( d \). From Lemma 3, the \((n - 1)\) paths \( Q_i \) \((2 \leq i \leq n)\) generated in Step 2 are disjoint other than \( d \) with each other. Because \( Q_i \) is outside of \( M^j \) other than \( d^{(1,i)} \), it is disjoint other than \( d^{(1,i)} \) from the paths \( P_i \) \((2 \leq i \leq n)\). Then the path \( R_i : s \rightarrow d^{(1,i)} \rightarrow d \) is also disjoint other than \( s \) and \( d \) with other paths \( R_i \) \((1 \leq i \neq h \leq n)\).

By hypothesis of induction, Step 1 takes \( T(n - 1) \) time to generate \((n - 1)\) paths and the maximum length of them is \( L(n - 1) \). From Lemma 3, Step 2 takes \( O(n) \) time to generate the \((n - 1)\) paths \( Q_i \) \((2 \leq i \leq n)\) of lengths at most \( 4 \). Thus, the length of the path \( R_i \) is at most \( 4 \), and the lengths of the paths \( R_i \) \((2 \leq i \neq h \leq n)\) are at most \( L(n - 1) + 3 = L(n - 1) + 1 \). Also, the length of the path \( R_h \) is 2. Consequently, Procedure 3 takes \( T(n - 1) + O(n) \) time to generate the \( n \) paths \( R_i \) \((1 \leq i \leq n)\) whose lengths are at most \( L(n - 1) + 3 \).

**Lemma 7:** For two nodes \( s \) and \( d \) in an \( M_n \), Procedure 4 generates \( n \) paths from \( s \) to \( d \) that are disjoint other than \( s \) and \( d \). The time complexity of Procedure 4 is \( T(n - 1) + O(n) \) and the maximum length of the paths generated is \( \max\{[n/2] + 5, L(n - 1) + 3\} \).

(Proof) The \((n - 1)\) paths \( P_i \) \((2 \leq i \leq n)\) generated in Step 1 are disjoint other than \( s \) and \( d \) by hypothesis of induction. From Lemma 3, the \((n - 1)\) paths \( Q_i \) \((2 \leq i \leq n)\) generated in Step 2 are disjoint other than \( d \) with each other. Because \( Q_i \) is outside of \( M^j \) other than \( d^{(1,i)} \), it is disjoint other than \( d^{(1,i)} \) from the paths \( P_i \) \((2 \leq i \leq n)\). The path \( s \rightarrow s^{(1)} \rightarrow u \rightarrow d \) is also disjoint other than \( s \) and \( d \) with other paths \( P_i \) and \( Q_i \) \((2 \leq i \neq h \leq n)\).

Step 1 takes \( T(n - 1) \) time to generate \((n - 1)\) paths and the maximum length of them is \( L(n - 1) \) by hypothesis of induction. From Lemma 3, Step 2 takes \( O(n) \) time to generate the \((n - 1)\) paths \( Q_i \) \((2 \leq i \leq n)\) of lengths at most \( 4 \). Step 3 takes \( O(1) \) time to choose two edges \( s \rightarrow s^{(1)} \) and \( d \rightarrow d^{(1)} \). It takes \( O(n) \) time in Step 4 to generate the path \( s^{(1)} \rightarrow d \) of length at most \( (n + 1)/2 \) \(= [n/2] + 1 \). In Step 5, we can find the node \( u \) as follows. Because each \( Q_i \) \((2 \leq i \leq n)\) has at most three internal nodes, it is enough to check each of the last three internal nodes of the path \( P_1 \): \( s^{(1)} \rightarrow d \) if it is \( u \) or not. If one of the other nodes on \( P_1 \) were \( u \), it would be incompatible with the fact that \( P_1 \) is the
shortest path from \(s^{(1)}\) to \(d\). Hence, we can find \(u\) in \(O(n)\) time. In Step 6, deleting the subpath and the \((n-2)\) edges takes \(O(n)\) time. The time complexity is \(T(n-1) + O(n) + O(1) + O(n)+ O(n) + O(n) = T(n-1) + O(n)\) in total. The length of the path \(R_1\) is at most \((n/2)+1)+3=\lfloor n/2\rfloor+5\). The maximum length of the paths \(R_i\) \((2 \leq i \leq n)\) is \((L(n-1)-1)+4 = L(n-1)+3\). The length of the path \(R_k\) is at most \(L(n-1)+1\). Therefore, the maximum path length is \(\max\{\lfloor n/2\rfloor+5, L(n-1)+3\}\).

\[ \text{Theorem 1:} \quad \text{For a node} \ s \ \text{and a node} \ d \ \text{in an} \ M_n \ \text{where} \ n \geq 4, \ \text{Algorithm N2N finds} \ n \ \text{paths from} \ s \ \text{to} \ d \ \text{that are disjoint other than} \ s \ \text{and} \ d. \ \text{The time complexity} \ T(n) \ \text{of} \ N2N \ \text{is} \ O(n^2), \ \text{and the maximum path length} \ L(n) = 3n - 5. \ \text{(Proof)} \ \text{From Lemmas 4 to 7, the generated paths are disjoint other than} \ s \ \text{and} \ d. \ \text{Also,} \ T(n) = O(n^2) \ \text{from} \ T(n) = T(n-1) + O(n) \ \text{and} \ T(2) = O(1). \ \text{From} \ L(n) = \max\{\lfloor n/2\rfloor+5, L(n-1)+3\} \ \text{and} \ L(3) = 4, \ \text{the equation} \ 3n - 5 \ \text{is derived}. \]

5. Conclusions

We proposed here a polynomial-order time algorithm for the node-to-node disjoint paths problem in \(n\)-Möbius cubes. Its time complexity is \(O(n^2)\) and the maximum path length is \(3n - 5\).

Future research includes theoretical analysis of the maximum path length of the algorithm, and improvement of the algorithm to generate shorter paths in smaller execution time. Designing an algorithm that solves the set-to-set disjoint paths problem in \(n\)-Möbius cubes is also interesting for us.

Acknowledgments

We express our special gratitude to Dr. Bipin Indurkhya for his proofreading. We also appreciate the anonymous reviewers for their insightful comments and constructive suggestions. This study was partly supported by a Grant-in-Aid for Scientific Research (C) of the Japan Society for the Promotion of Science under Grant No. 17K00993.

References

David Kocík is a master program student of Faculty of Information Technology at Czech Technical University in Prague in Czech Republic. His main research areas are graph theory, dependable computing, and fault-tolerant systems. He received the B.E. degree from Czech Technical University in Prague in 2013.

Keiichi Kaneko is a Professor at Tokyo University of Agriculture and Technology in Japan. His main research areas are functional programming, parallel and distributed computation, partial evaluation and fault-tolerant systems. He received the B.E., M.E. and Ph.D. degrees from the University of Tokyo in 1985, 1987 and 1994, respectively. He is a member of ACM, IEEE CS, IPSJ and JSSST.