PAPER

Node-to-Set Disjoint Paths Problem in Cross-Cubes

Rikuya SASAKI[†], **Hiroyuki ICHIDA**[†], **Htoo Htoo Sandi KYAW**[†], ***Nonmembers***, ***and*** **Keiichi KANEKO**[†a)], ***Member***

SUMMARY The increasing demand for high-performance computing in recent years has led to active research on massively parallel systems. The interconnection network in a massively parallel system interconnects hundreds of thousands of processing elements so that they can process large tasks while communicating among others. By regarding the processing elements as nodes and the links between processing elements as edges, respectively, we can discuss various problems of interconnection networks in the framework of the graph theory. Many topologies have been proposed for interconnection networks of massively parallel systems. The hypercube is a very popular topology and it has many variants. The cross-cube is such a topology, which can be obtained by adding one extra edge to each node of the hypercube. The cross-cube reduces the diameter of the hypercube, and allows cycles of odd lengths. Therefore, we focus on the cross-cube and propose an algorithm that constructs disjoint paths from a node to a set of nodes. We give a proof of correctness of the algorithm. Also, we show that the time complexity and the maximum path length of the algorithm are $O(n^3 \log n)$ and $2n - 3$, respectively. Moreover, we estimate that the average execution time of the algorithm is $O(n^2)$ based on a computer experiment.

key words: *high-performance computing, hypercube, interconnection network, massively parallel system, shortest path, topology*

1. Introduction

Today, the demand for high-performance computing prevails in many fields. However, the performance of a single processor hits a ceiling, and the large-scale computation becomes difficult because it takes a considerable amount of time to implement. Hence, much attention is focused on the parallel computation, which improves computation performance by interconnecting multiple processors through a network and distributing tasks to them.

A topology of the interconnection network of a parallel system decides the pattern to interconnect processing elements in the system. The system performance such as the communication bandwidth and the dependability significantly relies on the network topology. There are many topologies proposed for interconnection networks [1]–[9]. Among them, we focus on the cross-cube [3], which is a variant of the hypercube [6]. The hypercube has been adopted for various parallel systems. The cross-cube is obtained by adding an extra edge to each node of the hypercube. The diameter of an n-dimensional cross-cube is $n - 1$, which is smaller than the diameter of an n-dimensional hypercube by one. In addition, the cross-cube has cycles of odd lengths of 3 or more while the hypercube does not include any cycle with an odd length. Hence, the cross-cube has more flexibility than the hypercube.

The node-to-set disjoint paths problem is to find k paths P_i: $\boldsymbol{s} \rightsquigarrow \boldsymbol{d}_i$ $(1 \leq i \leq k)$ between a source node $\boldsymbol{s}$ and a set of destination nodes $\{\boldsymbol{d}_1, \boldsymbol{d}_2, \ldots, \boldsymbol{d}_k\}$ in a k-connected graph $G(V, E)$ so that the paths P_i are node-disjoint, or disjoint in short, except for $\boldsymbol{s}$. The node-to-set disjoint paths problem [10]–[17] is as important as the node-to-node disjoint paths problem [18]–[22] and the set-to-set disjoint paths problem [23]–[28]. If an algorithm can solve the node-to-set disjoint paths problem efficiently in a topology, it can be applied to improve the performance of the systems with networks based on the topology. That is, the system can maximize its communication bandwidth and communicate avoiding faulty nodes and/or links inside.

In this paper, we propose an algorithm that solves the node-to-set disjoint paths problem in a cross-cube. In an n-dimensional cross-cube, Wang et al. have proposed an algorithm that solves the node-to-set disjoint paths problem [29]. Their algorithm constructs $(n + 1)$ paths from a source node to $(n + 1)$ destination nodes that are disjoint except for the source node. It takes $O(n^2 2^n)$ time and the path lengths are at most $2n - 2$. As it is well-known, Menger's theorem ensures the existence of a solution [30], and we can obtain one of the solutions by applying a maximum-flow algorithm. Even if we apply the simple Ford-Fulkerson algorithm [31] in an n-dimensional cross-cube, we can obtain the paths from the source node to the destination nodes that are disjoint except for the source node in $O(n2^n)$ time. Therefore, the contribution of Wang et al. is that they have shown that the path lengths are at most $2n - 2$. The algorithm we propose in this paper solves the problem in $O(n^3 \log n)$ time with the maximum path length $2n - 3$. Hence, it provides significant improvement regarding the time complexity while decreasing the upper bound of the maximum path length by one.

The rest of this paper is organized as follows. Section 2 gives a definition of the cross-cube and a lemma regarding its property. Next, we propose an algorithm, SPR (Shortest-Path Routing), that constructs one of the shortest paths between an arbitrary pair of nodes in the cross-cube in Sect. 3. Then, we give an algorithm that constructs disjoint paths between a source node and a set of destination nodes in the cross-cube in Sect. 4. In Sect. 5, we give the proof of correctness of the algorithm and its time complexity. We

Manuscript received April 1, 2023.
Manuscript revised July 7, 2023.
Manuscript publicized October 6, 2023.

[†]The authors are with Graduate School of Engineering, Tokyo University of Agriculture and Technology, Koganei-shi, 184–8588 Japan.
a) E-mail: k1kaneko@cc.tuat.ac.jp (Corresponding author)

DOI: 10.1587/transinf.2023EDP7067

also estimate the maximum length of the constructed paths. Section 6 shows the result of a computer experiment. Finally, Sect. 7 gives the conclusion and the future work of this study.

2. Preliminaries

In this section, we give a definition of the cross-cube and a lemma related to it following the notations and terminology from the traditional graph theory. For example, a path in a graph $G(V, E)$ is an alternate sequence of nodes and edges: $\boldsymbol{a}_1, (\boldsymbol{a}_1, \boldsymbol{a}_2), \boldsymbol{a}_2, \ldots, \boldsymbol{a}_{l-1}, (\boldsymbol{a}_{l-1}, \boldsymbol{a}_l), \boldsymbol{a}_l$ for $\boldsymbol{a}_i \in V$ $(1 \leq i \leq l)$, and we use a shorthand $\boldsymbol{a}_1 \rightarrow \boldsymbol{a}_2 \rightarrow \cdots \rightarrow \boldsymbol{a}_l$ or $\boldsymbol{a}_1 \rightsquigarrow \boldsymbol{a}_l$. The length of a path is the number of edges included in the path. For two nodes $\boldsymbol{a}$ and $\boldsymbol{b}$, the paths between them with the shortest length are called the shortest paths, and the distance between them, $d(\boldsymbol{a}, \boldsymbol{b})$, is given by the length of the shortest paths between them. For two paths L_0: $\boldsymbol{a}_1 \rightsquigarrow \boldsymbol{a}_l$ and L_1: $\boldsymbol{b}_1 \rightsquigarrow \boldsymbol{b}_m$, they are disjoint if they do not have any common node. If L_0 and L_1 do not have any common node except for their terminal nodes $\boldsymbol{a}_1 = \boldsymbol{b}_1$, they are disjoint except for $\boldsymbol{a}_1 (= \boldsymbol{b}_1)$.

Definition 1: An n-dimensional cross-cube C_n $(n \geq 2)$ is an undirected graph whose node set, $V(C_n)$, is $\{0,1\}^n$. For two nodes $\boldsymbol{a} = (a_1, a_2, \ldots, a_n)$ and $\boldsymbol{b} = (b_1, b_2, \ldots, b_n) (\in V(C_n))$, an edge $(\boldsymbol{a}, \boldsymbol{b}) (\in E(C_n))$ exists if and only if either $H(\boldsymbol{a}, \boldsymbol{b}) = 1$ or $a_i = b_i$ $(1 \leq i \leq n-2)$, $a_{n-1} = \overline{b_{n-1}}$, and $a_n = \overline{b_n}$ holds. Note that $H(\boldsymbol{a}, \boldsymbol{b})$ represents the Hamming distance between $\boldsymbol{a}$ and $\boldsymbol{b}$, and it is calculated by $H(\boldsymbol{a}, \boldsymbol{b}) = \sum_{i=1}^{n} a_i \oplus b_i$, where $\oplus$ is the exclusive-or operator defined by $0 \oplus 0 = 1 \oplus 1 = 0$ and $0 \oplus 1 = 1 \oplus 0 = 1$. Then, the neighbor node $\boldsymbol{b}$ obtained by reverting the ith bit $(1 \leq i \leq n)$ of $\boldsymbol{a}$ is represented by $\boldsymbol{a}^{(i)}$, and the neighbor node $\boldsymbol{b}$ obtained by reverting the $(n-1)$th and nth bits of $\boldsymbol{a}$ simultaneously is represented by $\boldsymbol{a}^{\mathrm{c}}$. In addition, the node obtained by reverting the jth bit $(1 \leq j \leq n)$ of a node $\boldsymbol{a}^{(i)}$ $(1 \leq i \leq n)$ is represented by $\boldsymbol{a}^{(i,j)}$.

Figure 1 shows an example of a 4-dimensional cross-cube, C_4.

Let $C_n^{[i:b]}$ represent the sub graph of C_n induced by the set of nodes $\boldsymbol{a} = (a_1, a_2, \ldots, a_n) (\in V(C_n))$ with $a_i = b$ $(1 \leq i \leq n, b \in \{0,1\})$.

Lemma 1: For a node $\boldsymbol{a}$ in $V(C_n^{[1:b]})$ $(n \geq 3, b \in \{0,1\})$, we can construct $(n+1)$ paths of lengths at most 2 from $\boldsymbol{a}$ to $\boldsymbol{a}^{(1)}, \boldsymbol{a}^{(2,1)}, \ldots, \boldsymbol{a}^{(n,1)}$, and $(\boldsymbol{a}^{\mathrm{c}})^{(1)}$ in $V(C_n^{[1:\overline{b}]})$ such that the paths are disjoint except for $\boldsymbol{a}$.

(Proof) We can construct $(n+1)$ paths, L_i $(1 \leq i \leq n+1)$, of lengths at most 2 as follows (Fig. 2):

$$L_i : \begin{cases} \boldsymbol{a} \rightarrow \boldsymbol{a}^{(1)} & (i = 1) \\ \boldsymbol{a} \rightarrow \boldsymbol{a}^{(i)} \rightarrow \boldsymbol{a}^{(i,1)} & (2 \leq i \leq n) \\ \boldsymbol{a} \rightarrow \boldsymbol{a}^{\mathrm{c}} \rightarrow (\boldsymbol{a}^{\mathrm{c}})^{(1)} & (i = n+1) \end{cases}$$

Then, $\boldsymbol{a}^{(1)} \in V(C_n^{[1:\overline{b}]})$ and $H(\boldsymbol{a}, \boldsymbol{a}^{(1)}) = 1$ hold. On the other hand, because $H(\boldsymbol{a}, \boldsymbol{a}^{(i,1)}) = 2$ $(2 \leq i \leq n)$ and $H(\boldsymbol{a}, (\boldsymbol{a}^{\mathrm{c}})^{(1)}) = 3$, $\boldsymbol{a}^{(1)}$ is not included in any other paths. Hence, L_1 is disjoint from other paths except for $\boldsymbol{a}$. Also, $(\boldsymbol{a}^{\mathrm{c}})^{(1)}$ and $\boldsymbol{a}^{(i,1)}$ $(2 \leq i \leq n)$ are distinct. Moreover, because $H(\boldsymbol{a}, \boldsymbol{a}^{\mathrm{c}}) = 2$ and $H(\boldsymbol{a}, \boldsymbol{a}^{(i)}) = 1$ $(2 \leq i \leq n)$, $\boldsymbol{a}^{\mathrm{c}}$ and $\boldsymbol{a}^{(i)}$ $(2 \leq i \leq n)$ are distinct. Hence, L_{n+1} is disjoint from L_i $(2 \leq i \leq n)$ except for $\boldsymbol{a}$. Furthermore, for i and j such that $2 \leq i \neq j \leq n$, $\boldsymbol{a}^{(i)}$ and $\boldsymbol{a}^{(j)}$ are distinct. Also, $\boldsymbol{a}^{(i,1)}$ and $\boldsymbol{a}^{(j,1)}$ are distinct. Hence, L_i and L_j are disjoint except for $\boldsymbol{a}$. From the above discussion, the $(n+1)$ paths, L_i $(1 \leq i \leq n+1)$, of lengths at most 2 are disjoint except for $\boldsymbol{a}$. □

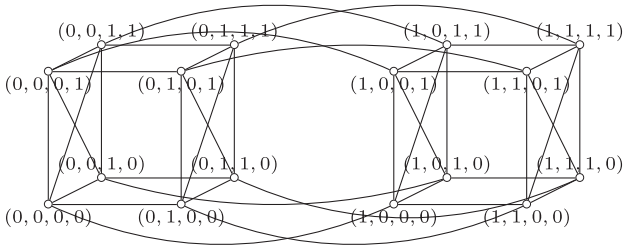


Fig. 1 Example of a 4-dimensional cross-cube, C_4.

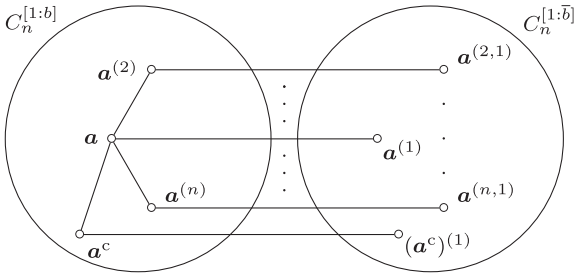


Fig. 2 $(n+1)$ disjoint paths from node $\boldsymbol{a} (\in V(C_n^{[1:b]}))$ to $C_n^{[1:\overline{b}]}$.

3. SPR Algorithm

For a source node $\boldsymbol{s} = (s_1, s_2, \ldots, s_n)$ and a destination node $\boldsymbol{d} = (d_1, d_2, \ldots, d_n)$ in $V(C_n)$, we show an algorithm SPR in this section that constructs one of the shortest paths of length at most $n-1$ between $\boldsymbol{s}$ and $\boldsymbol{d}$.

Step 1 If $\boldsymbol{s} = \boldsymbol{d}$, terminate.
Step 2 Let $i^* = \min\{i \mid s_i \neq d_i, 1 \leq i \leq n\}$.
Step 3 If $i^* \leq n-2$, let $\boldsymbol{s}$ be $\boldsymbol{s}^{(i^*)}$, and go back to Step 1.
Step 4 If $i^* = n$, let $\boldsymbol{s}$ be $\boldsymbol{s}^{(n)}$, and terminate.
Step 5 If $s_n = d_n$, let $\boldsymbol{s}$ be $\boldsymbol{s}^{(n-1)}$, and terminate.
Step 6 Let $\boldsymbol{s}$ be $\boldsymbol{s}^{\mathrm{c}}$, and terminate.

SPR Algorithm clearly constructs one of the shortest paths of length at most $n-1$ between $\boldsymbol{s}$ and $\boldsymbol{d}$ in $O(n)$ time.

4. N2S-R Algorithm

For a source node $\boldsymbol{s}$ and a set of destination nodes $D = \{\boldsymbol{d}_1, \boldsymbol{d}_2, \ldots, \boldsymbol{d}_{n+1}\}$ in $V(C_n)$, we give an algorithm, N2S-R, that constructs $(n+1)$ paths that are disjoint except for $\boldsymbol{s}$. If $n = 2$, the three disjoint paths are trivially $\boldsymbol{s} \rightarrow \boldsymbol{s}^{(1)}$, $\boldsymbol{s} \rightarrow \boldsymbol{s}^{(2)}$, and $\boldsymbol{s} \rightarrow \boldsymbol{s}^{\mathrm{c}}$. Hence, we assume that $n \geq 3$ in the rest of

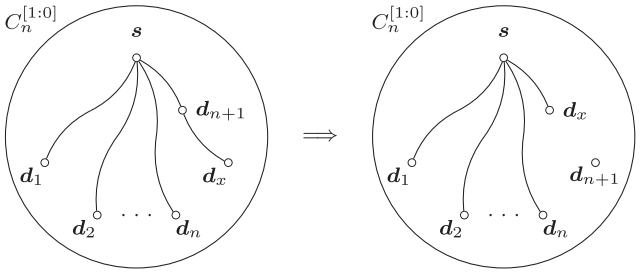


Fig. 3 After Step 2 of Case 1.

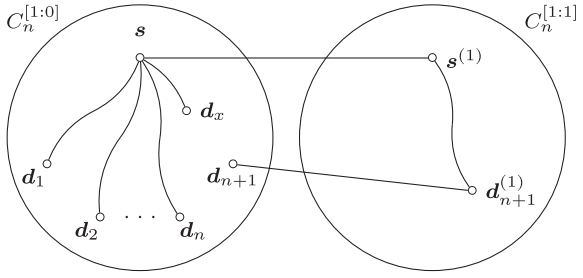


Fig. 4 $(n+1)$ disjoint paths U_i: $s \rightsquigarrow d_i$ $(1 \le i \le n+1)$ in Case 1.

this paper. N2S-R Algorithm consists of three procedures depending on the relative distribution of the source node and the destination nodes. Without loss of generality, we can assume that $s \in V(C_n^{[1:0]})$.

4.1 Case 1 ($|D \cap V(C_n^{[1:0]})| = n+1$)

In this case, we can construct $(n+1)$ disjoint paths from s to D, U_i: $s \rightsquigarrow d_i$ $(1 \le i \le n+1)$, by using Procedure 1 below.

Procedure 1

Step 1 In $C_n^{[1:0]}$, apply N2S-R Algorithm recursively to construct n disjoint paths, P_i: $s \rightsquigarrow d_i$ $(1 \le i \le n)$, from s to $d_1, d_2, \ldots, d_n$.

Step 2 If d_{n+1} is included in one of P_i, say P_x, discard the sub path $d_{n+1} \rightsquigarrow d_x$ to obtain P_x: $s \rightsquigarrow d_{n+1}$, and exchange the indices of d_x and d_{n+1} (Fig. 3).

Step 3 Select two edges $s \to s^{(1)}$ and $d_{n+1}^{(1)} \to d_{n+1}$.

Step 4 In $C_n^{[1:1]}$, apply SPR Algorithm to construct one of the shortest paths, P_{n+1}: $s^{(1)} \rightsquigarrow d_{n+1}^{(1)}$, from $s^{(1)}$ to $d_{n+1}^{(1)}$.

Step 5 Construct $(n+1)$ disjoint paths, U_i: $s \rightsquigarrow d_i$ $(1 \le i \le n+1)$, as follows (Fig. 4):

$$U_i : \begin{cases} s \overset{P_i}{\rightsquigarrow} d_i & (1 \le i \le n) \\ s \to s^{(1)} \overset{P_{n+1}}{\rightsquigarrow} d_{n+1}^{(1)} \to d_{n+1} & (i = n+1) \end{cases}$$

4.2 Case 2 ($|D \cap V(C_n^{[1:0]})| = n$)

In this case, we can assume without loss of generality that $D \cap V(C_n^{[1:0]}) = \{d_1, d_2, \ldots, d_n\}$ and $D \cap V(C_n^{[1:1]}) = \{d_{n+1}\}$.

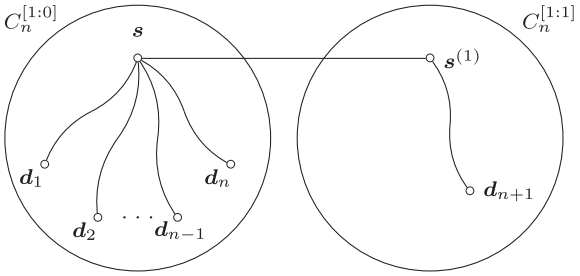


Fig. 5 $(n+1)$ disjoint paths U_i: $s \rightsquigarrow d_i$ $(1 \le i \le n+1)$ in Case 2.

Then, we can construct $(n+1)$ disjoint paths from s to D, U_i: $s \rightsquigarrow d_i$ $(1 \le i \le n+1)$, by using Procedure 2 below.

Procedure 2

Step 1 In $C_n^{[1:0]}$, apply N2S-R Algorithm recursively to construct n disjoint paths, P_i: $s \rightsquigarrow d_i$ $(1 \le i \le n)$ from s to $d_1, d_2, \ldots, d_n$.

Step 2 Select the edge $s \to s^{(1)}$.

Step 3 In $C_n^{[1:1]}$, apply SPR Algorithm to construct one of the shortest paths, P_{n+1}: $s^{(1)} \rightsquigarrow d_{n+1}$, from $s^{(1)}$ to d_{n+1}.

Step 4 Construct $(n+1)$ disjoint paths, U_i: $s \rightsquigarrow d_i$ $(1 \le i \le n+1)$, as follows (Fig. 5):

$$U_i : \begin{cases} s \overset{P_i}{\rightsquigarrow} d_i & (1 \le i \le n) \\ s \to s^{(1)} \overset{P_{n+1}}{\rightsquigarrow} d_{n+1} & (i = n+1) \end{cases}$$

4.3 Case 3 ($|D \cap V(C_n^{[1:0]})| = k < n$)

In this case, we can assume that without loss of generality that $D \cap V(C_n^{[1:0]}) = \{d_1, d_2, \ldots, d_k\}$ and $D \cap V(C_n^{[1:1]}) = \{d_{k+1}, d_{k+2}, \ldots, d_{n+1}\}$ $(k < n)$. Then, we can construct $(n+1)$ disjoint paths from s to D, U_i: $s \rightsquigarrow d_i$ $(1 \le i \le n+1)$, by using Procedure 3 below.

Procedure 3

Step 1 For each of the destination nodes d_i $(k+1 \le i \le n+1)$, if $d_i^{(1)} \notin D$, select the edge $d_i^{(1)} \to d_i$, and let it be the path P_i: $d_i^{(1)} \to d_i$. Also, let d_i' be $d_i^{(1)}$. Without loss of generality, we can assume that edges $d_i^{(1)} (= d_i') \to d_i$ $(k+1 \le i \le l)$ were selected.

Step 2 For each of the destination nodes d_i $(l+1 \le i \le n+1)$, consider $(n-1)$ paths, $d_i^{(j,1)} \to d_i^{(j)} \to d_i$ $(2 \le j \le n)$. If there is a path among them such that it does not include any other destination nodes or the nodes on the paths constructed from the other destination nodes, select it as P_i: $d_i' \rightsquigarrow d_i$. Without loss of generality, we can assume that paths P_i: $d_i' \rightsquigarrow d_i$ $(l+1 \le i \le m)$ were selected.

Step 3 If $m = n$, select the path P_{n+1}: $(d_{n+1}{}^{\mathrm{c}})^{(1)} \to d_{n+1}{}^{\mathrm{c}} \to d_{n+1}$, and let d_{n+1}' be $(d_{n+1}{}^{\mathrm{c}})^{(1)}$ (Fig. 6).

Step 4 Select the edge $s \to s^{(1)}$.

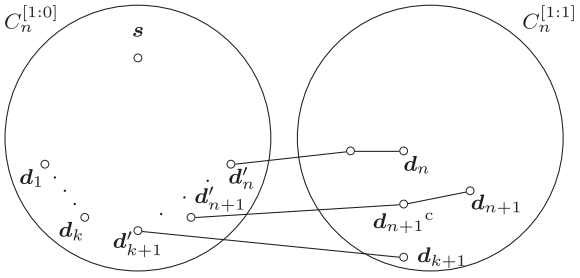


Fig. 6 After Step 3 of Case 3 with $m = n$.

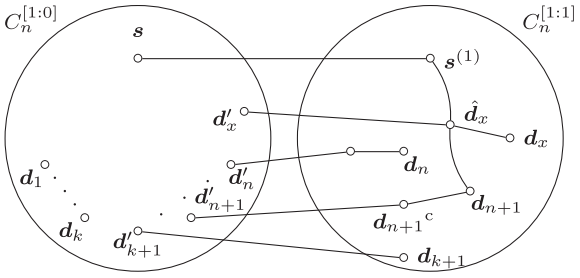


Fig. 7 During Step 6 of Case 3.

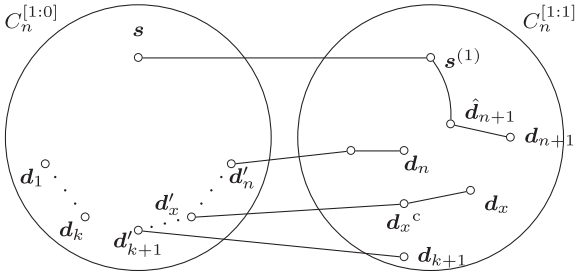


Fig. 8 After Step 7 of Case 3.

Step 5 In $C_n^{[1:1]}$, apply SPR Algorithm to construct one of the shortest paths, Q: $\boldsymbol{s}^{(1)} \rightsquigarrow \boldsymbol{d}_{n+1}$, from the node $\boldsymbol{s}^{(1)}$ to the destination node $\boldsymbol{d}_{n+1}$.

Step 6 If the path Q includes some of the nodes on the path constructed in Steps 1 to 3, let $\hat{\boldsymbol{d}}_x$ be the closest one to $\boldsymbol{s}^{(1)}$ along Q. Also, let P_x: $\boldsymbol{d}'_x \rightsquigarrow \hat{\boldsymbol{d}}_x \rightsquigarrow \boldsymbol{d}_x$ be the path to which $\hat{\boldsymbol{d}}_x$ belongs (Fig. 7). Then, discard the sub path $\hat{\boldsymbol{d}}_x \rightsquigarrow \boldsymbol{d}_{n+1}$ of Q, and select Q: $\boldsymbol{s}^{(1)} \rightsquigarrow \hat{\boldsymbol{d}}_x \rightsquigarrow \boldsymbol{d}_x$. Also, exchange the indices of P_x and P_{n+1}, the indices of $\boldsymbol{d}_x$ and $\boldsymbol{d}_{n+1}$, and the indices of $\boldsymbol{d}'_x$ and $\boldsymbol{d}'_{n+1}$.

Step 7 Discard the path P_{n+1}: $\boldsymbol{d}'_{n+1} \rightsquigarrow \boldsymbol{d}_{n+1}$ (Fig. 8).

Step 8 In $C_n^{[1:0]}$, apply N2S-R Algorithm recursively to construct paths R_i: $\boldsymbol{s} \rightsquigarrow \boldsymbol{d}_i$ $(1 \le i \le k)$ and $\boldsymbol{s} \rightsquigarrow \boldsymbol{d}'_i$ $(k+1 \le i \le n)$, from the source node $\boldsymbol{s}$ to $\boldsymbol{d}_1, \boldsymbol{d}_2, \ldots, \boldsymbol{d}_k, \boldsymbol{d}'_{k+1}, \boldsymbol{d}'_{k+2}, \ldots, \boldsymbol{d}'_n$ that are disjoint except for $\boldsymbol{s}$.

Step 9 Construct $(n+1)$ disjoint paths, U_i: $\boldsymbol{s} \rightsquigarrow \boldsymbol{d}_i$ $(1 \le i \le n+1)$, as follows (Fig. 9):

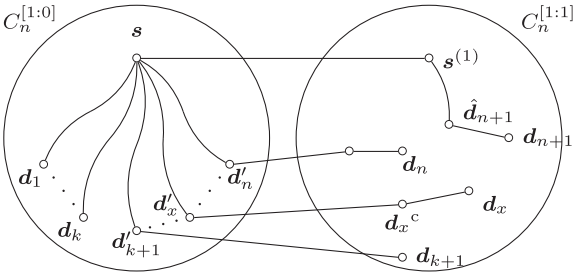


Fig. 9 $(n+1)$ disjoint paths U_i: $\boldsymbol{s} \rightsquigarrow \boldsymbol{d}_i$ $(1 \le i \le n+1)$ in Case 3.

$$U_i : \begin{cases} \boldsymbol{s} \overset{R_i}{\rightsquigarrow} \boldsymbol{d}_i & (1 \le i \le k) \\ \boldsymbol{s} \overset{R_i}{\rightsquigarrow} \boldsymbol{d}'_i \overset{P_i}{\rightsquigarrow} \boldsymbol{d}_i & (k+1 \le i \le n) \\ \boldsymbol{s} \to \boldsymbol{s}^{(1)} \overset{Q}{\rightsquigarrow} \boldsymbol{d}_{n+1} & (i = n+1) \end{cases}$$

5. Correctness and Complexities

Let $T(n)$ and $L(n)$ represent the time complexity and the upper bound of the lengths of the paths constructed by N2S-R Algorithm applied to C_n.

Lemma 2: Procedure 1 constructs $(n+1)$ disjoint paths of lengths at most $\max\{L(n-1), n\}$ from $\boldsymbol{s}$ to D in $T(n-1) + O(n \times L(n-1))$ time.

(Proof) From the hypothesis of induction, U_i $(1 \le i \le n)$ are disjoint except for $\boldsymbol{s}$. Also, from Step 2, U_i $(1 \le i \le n)$ do not include $\boldsymbol{d}_{n+1}$. In addition, because U_{n+1} is included in $C_n^{[1:1]}$ except for $\boldsymbol{s}$ and $\boldsymbol{d}_{n+1}$, it is disjoint from other U_i $(1 \le i \le n)$ except for $\boldsymbol{s}$.

Step 1 takes $T(n-1)$ time to construct n paths of lengths at most $L(n-1)$. Step 2 takes $O(n \times L(n-1))$ time to check if $\boldsymbol{d}_{n+1}$ is included in the paths constructed in Step 1. If it is included, it takes $O(1)$ time to discard the sub path $\boldsymbol{d}_{n+1} \rightsquigarrow \boldsymbol{d}_x$ and exchange the indices. Step 3 takes $O(1)$ time to select two edges of length 1. Step 4 takes $O(n)$ time to construct a path of length at most $n-2$. From the above discussion, Procedure 1 constructs $(n+1)$ disjoint paths of lengths at most $\max\{L(n-1), n\}$ in $T(n-1) + O(n \times L(n-1))$ time. □

Lemma 3: Procedure 2 constructs $(n+1)$ disjoint paths of lengths at most $\max\{L(n-1), n-1\}$ from $\boldsymbol{s}$ to D in $T(n-1) + O(n)$ time.

(Proof) From the hypothesis of induction, U_i $(1 \le i \le n)$ are disjoint except for $\boldsymbol{s}$. Also, because U_{n+1} is included in $C_n^{[1:1]}$ except for $\boldsymbol{s}$, it is disjoint from other U_i $(1 \le i \le n)$ except for $\boldsymbol{s}$.

Step 1 takes $T(n-1)$ time to construct n paths of lengths at most $L(n-1)$. Step 2 takes $O(1)$ time to select an edge of length 1. Step 3 takes $O(n)$ time to construct a path of length at most $n-2$. From the above discussion, Procedure 2 constructs $(n+1)$ disjoint paths of lengths at most $\max\{L(n-1), n-1\}$ in $T(n-1) + O(n)$ time. □

Lemma 4: Procedure 3 constructs $(n+1)$ disjoint paths of lengths at most $\max\{L(n-1)+2, n-1\}$ from $\boldsymbol{s}$ to D in $T(n-1) + O(n^2 \log n)$ time.

(Proof) Step 1 selects disjoint edges P_i: $\boldsymbol{d}_i^{(1)} \to \boldsymbol{d}_i$ $(k+1 \le i \le l)$. Step 2 selects disjoint paths P_i: $\boldsymbol{d}_i' \rightsquigarrow \boldsymbol{d}_i$ $(l+1 \le i \le m)$. In Steps 1 and 2, if a path from $\boldsymbol{d}_{n+1}$ is not selected, it means that $\boldsymbol{d}_{n+1}^{(1)} \in D$. Also, in Step 2, if the path P_{i_1}: $\boldsymbol{d}_{i_1}^{(j_1,1)} \to \boldsymbol{d}_{i_1}^{(j_1)} \to \boldsymbol{d}_{i_1}$ is selected for a certain destination node $\boldsymbol{d}_{i_1}$ $(l+1 \le i_1 \le m)$, it means that $\boldsymbol{d}_{i_1}^{(1)} \in D$. Therefore, for a destination node $\boldsymbol{d}_{i_2}$ $(i_1+1 \le i_2 \le n)$, $\boldsymbol{d}_{n+1}$ blocks at most one of the n paths $\boldsymbol{d}_{i_2}^{(1)} \to \boldsymbol{d}_{i_2}$ and $\boldsymbol{d}_{i_2}^{(j,1)} \to \boldsymbol{d}_{i_2}^{(j)} \to \boldsymbol{d}_{i_2}$ $(2 \le j \le n)$ that are disjoint except for $\boldsymbol{d}_{i_2}$. However, $\boldsymbol{d}_{n+1}^{(1)} (\in D)$ cannot block any other remaining path. On the other hand, the path P_{i_1} blocks at most 2 of the n paths. However, $\boldsymbol{d}_{i_1}^{(1)} (\in D)$ cannot block any other remaining path. Hence, among n paths considered with the source node $\boldsymbol{d}_{i_2}$, one path can be always selected. In Step 3, for $\boldsymbol{d}_{n+1}$, if all of the n paths considered in Steps 1 and 2 cannot be selected, P_{n+1}: $(\boldsymbol{d}_{n+1}{}^{\mathrm{c}})^{(1)} \to \boldsymbol{d}_{n+1}{}^{\mathrm{c}} \to \boldsymbol{d}_{n+1}$ can be selected as a disjoint path from the above discussion and Lemma 1. The path $\boldsymbol{s} \to \boldsymbol{s}^{(1)} \rightsquigarrow \boldsymbol{d}_{n+1}$ constructed in Steps 4, 5, 6, and 7 is disjoint from other paths from the process of its construction. From the hypothesis of induction, R_i $(1 \le i \le n)$ are disjoint. Also, the paths are disjoint from the paths P_i $(k+1 \le i \le n)$ except for $\boldsymbol{d}_i'$. In addition, they are disjoint from the path $\boldsymbol{s} \to \boldsymbol{s}^{(1)} \rightsquigarrow \boldsymbol{d}_{n+1}$ except for $\boldsymbol{s}$. Hence, U_i $(1 \le i \le n+1)$ are disjoint except for $\boldsymbol{s}$.

In Step 1, for each destination node $\boldsymbol{d}_i$ $(k+1 \le i \le n+1)$, it takes $O(n)$ time to check $\boldsymbol{d}_i^{(1)} \notin D$ and select an edge of length 1. Hence, it takes $O(n^2)$ time in total. In Step 2, for each destination node $\boldsymbol{d}_i$ $(l+1 \le i \le n+1)$, by using the data structure of the balanced binary tree, it takes $O(n \log n)$ time to find one of the paths $\boldsymbol{d}_i^{(j,1)} \to \boldsymbol{d}_i^{(j)} \to \boldsymbol{d}_i$ $(2 \le j \le n)$ such that it does not include other destination nodes or the nodes on the paths constructed from other destination nodes, and select the path of length 2. Thus, it takes $O(n^2 \log n)$ time in total. Step 3 takes $O(1)$ time to select a path of length 2. Step 4 takes $O(1)$ time to select an edge. Step 5 takes $O(n)$ time to construct a path of length at most $n-2$. In Step 6, it takes $O(n^2)$ time to check the existence of $\hat{\boldsymbol{d}}_x$. If it exists, it takes $O(1)$ time to delete the sub path of Q, update Q, and exchange the indices. Step 7 takes $O(1)$ time to delete the path P_{n+1}. From the induction hypothesis, Step 8 takes $T(n-1)$ time to construct n paths of lengths at most $L(n-1)$ that are disjoint except for $\boldsymbol{s}$. From the above discussion, Procedure 3 constructs $(n+1)$ disjoint paths of lengths at most $\max\{L(n-1)+2, n-1\}$ in $T(n-1)+O(n^2 \log n)$ time. □

Theorem 1: For a source node $\boldsymbol{s}$ and a set of $(n+1)$ destination nodes D in $V(C_n)$, N2S-R Algorithm constructs $(n+1)$ paths of lengths at most $2n-3$ from $\boldsymbol{s}$ to D that are disjoint except for $\boldsymbol{s}$ in $O(n^3 \log n)$ time.

(Proof) From Lemmas 2 to 4, N2S-R Algorithm constructs $(n+1)$ paths from $\boldsymbol{s}$ to D that are disjoint except for $\boldsymbol{s}$. Also, from Lemmas 2 to 4, $T(n) = \max\{T(n-1) + O(n \times L(n-1)), T(n-1) + O(n^2 \log n)\}$ and $L(n) = \max\{L(n-1)+2, n\}$ hold. From $L(2) = 1$, $L(n) = 2n-3$ holds. Thus, $T(n) = T(n-1) + O(n^2 \log n) = O(n^3 \log n)$ holds. □

6. Computer Experiment

To observe the average behavior of N2S-R Algorithm, we have implemented it by using the programming language C++. We have conducted an experiment on a computer with the Intel Core i9-13900KS (3.20GHz) processor, 128.0GB memory, and the Microsoft Windows 11 Pro Edition operating system.

By taking advantage of the node-symmetric property of the cross-cube, the source node was fixed to $(0,0,\ldots,0)$, and the experiment was carried out in three steps as follows:

Step 1) For each n between 2 to 31, execute Steps 2 and 3 for 10,000,000 times.

Step 2) In an n-dimensional cross-cube, select $(n+1)$ distinct destination nodes randomly.

Step 3) Apply N2S-R, and measure the sum of execution time and the maximum path length.

Figures 10 and 11 show the average execution time and the maximum path length, respectively.

The obtained result regarding the time complexity (see Fig. 10) indicates that the average time complexity of the proposed algorithm is $O(n^2)$ except for the base case $n = 2$. This is to be compared with the theoretical worst-case time complexity that has been previously established (see Theorem 1). The gap between the pessimistic worst-case time complexity

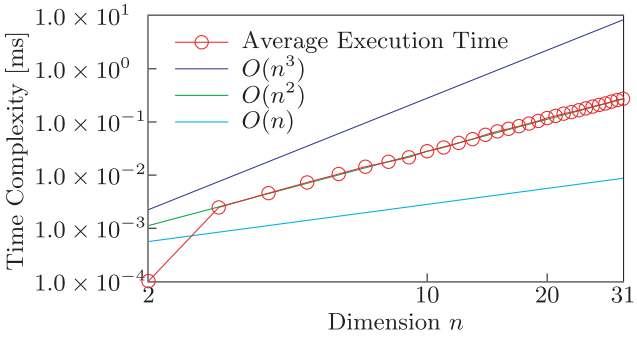


Fig. 10 Average time complexity of N2S-R Algorithm to construct $(n+1)$ disjoint paths in n-dimensional cross-cubes.

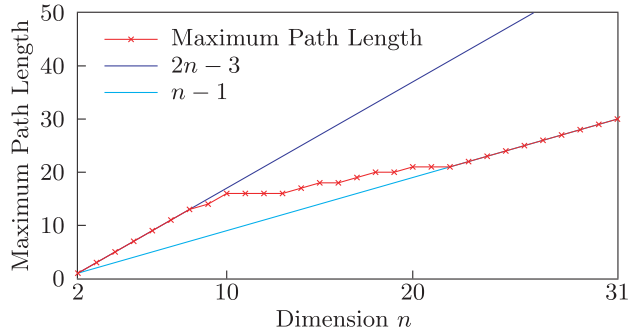


Fig. 11 Maximum length of $(n+1)$ disjoint paths constructed by N2S-R Algorithm in n-dimensional cross-cubes.

$O(n^3 \log n)$ and the experimentally estimated average time complexity $O(n^2)$ is a good indicator of the performance of the algorithm.

The experimental result (see Fig. 11) shows that the theoretical upper bound $2n-3$ is attained with $2 \leq n \leq 8$. But the maximum path length stays well on $n-1$ as the dimension n of the network increases, where $n-1$ is the diameter of the n-dimensional cross-cube and is also the lower bound of the maximum path length. So, the difference between the experimentally obtained maximum path length and the theoretical one is yet another strong indicator of the performance of our proposed algorithm.

7. Conclusion

In this paper, we have proposed an algorithm that solves the node-to-set disjoint paths problem in the cross-cube. For a source node $\boldsymbol{s}$ and a set of destination nodes $\{\boldsymbol{d}_1, \boldsymbol{d}_2, \ldots, \boldsymbol{d}_{n+1}\}$ in an n-dimensional cross-cube, our algorithm constructs $(n+1)$ paths U_i: $\boldsymbol{s} \rightsquigarrow \boldsymbol{d}_i$ $(1 \leq i \leq n+1)$ of lengths at most $2n-3$ such that the paths are disjoint except for $\boldsymbol{s}$ in $O(n^3 \log n)$ time. This is a significant improvement over the time complexity $O(n^2 2^n)$ of the algorithm by Wang et al. [29]. Also, we have conducted a computer experiment, which indicated that the average time complexity of our algorithm is $O(n^2)$. Also, it showed that the maximum path length stays well on $n-1$ with larger n, and the theoretical upper bound $2n-3$ is attained with $2 \leq n \leq 8$.

To reduce the time complexity of the algorithm by eliminating its recursive structure is one of our future works.

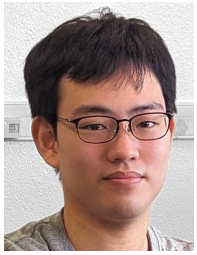
Acknowledgments

The authors would like to express special thanks to the reviewers for their insightful comments and 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. 23K11029.

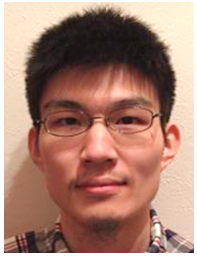
References

[1] P. Cull and S.M. Larson, "The Möbius cubes," IEEE Trans. Comput., vol.44, no.5, pp.647–659, May 1995.
[2] K. Efe, "The crossed cube architecture for parallel computation," IEEE Trans. Parallel Distrib. Syst., vol.3, no.5, pp.513–524, Sept. 1992.
[3] E. Haq, "Cross-cube: a new fault tolerant hypercube-based network," Proc. Fifth International Parallel Processing Symposium, pp.471–474, 1991.
[4] P.A.J. Hilbers, M.R.J. Koopman, and J.L.A. van de Snepscheut, "The twisted cube," Volume I: Parallel Architectures on PARLE: Parallel Architectures and Languages Europe, London, UK, pp.152–159, Springer-Verlag, 1987.
[5] H.-S. Lim, J.-H. Park, and H.-C. Kim, "The bicube: An interconnection of two hypercubes," International Journal of Computer Mathematics, vol.92, no.1, pp.29–40, Jan. 2015.
[6] C.L. Seitz, "The cosmic cube," Communications of the ACM, vol.28, no.1, pp.22–33, Jan. 1985.
[7] X. Wang, J. Liang, D. Qi, and W. Lin, "The twisted crossed cube," Concurrency and Computation: Practice and Experience, vol.28, no.5, pp.1507–1526, 2016.
[8] X. Yang, D.J. Evans, and G.M. Megson, "The locally twisted cubes," International Journal of Computer Mathematics, vol.82, no.4, pp.401–413, April 2005.
[9] W. Zhou, J. Fan, X. Jia, and S. Zhang, "The spined cube: a new hypercube variant with smaller diameter," Information Processing Letters, vol.111, no.12, pp.561–567, June 2011.
[10] A. Bossard and K. Kaneko, "A node-to-set disjoint paths routing algorithm in torus-connected cycles," ISCA International Journal of Computers and their Applications, vol.22, no.1, pp.22–30, Jan. 2015.
[11] Q.-P. Gu and S. Peng, "Node-to-set disjoint paths problem in star graphs," Information Processing Letters, vol.62, no.4, pp.201–207, April 1997.
[12] K. Kaneko and Y. Suzuki, "An algorithm for node-to-set disjoint paths problem in rotator graphs," IEICE Trans. Inf. & Syst., vol.E84-D, no.9, pp.1155–1163, Sept. 2001.
[13] K. Kaneko, "An algorithm for node-to-set disjoint paths problem in burnt pancake graphs," IEICE Trans. Inf. & Syst., vol.E86-D, no.12, pp.2588–2594, Dec. 2003.
[14] 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, March 2016.
[15] C.-N. Lai, "An efficient construction of one-to-many node-disjoint paths in folded hypercubes," Journal of Parallel and Distributed Computing, vol.74, no.4, pp.2310–2316, April 2014.
[16] 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.
[17] L. Lipták, E. Cheng, J.-S. Kim, and S.W. Kim, "One-to-many node-disjoint paths of hyper-star networks," Discrete Applied Mathematics, vol.160, no.13-14, pp.2006–2014, Sept. 2012.
[18] K. Kaneko and N. Sawada, "An algorithm for node-to-node disjoint paths problem in burnt pancake graphs," IEICE Trans. Inf. & Syst., vol.E90-D, no.1, pp.306–313, Jan. 2007.
[19] D. Kocík and K. Kaneko, "Node-to-node disjoint paths problem in a Möbius cubes," IEICE Trans. Inf. & Syst., vol.E100-D, no.8, pp.1837–1843, Aug. 2017.
[20] T.-C. Lin and D.-R. Duh, "Constructing vertex-disjoint paths in (n,k)-star graphs," Information Sciences, vol.178, no.3, pp.788–801, Feb. 2008.
[21] Y. Li, S. Peng, and W. Chu, "Disjoint-paths and fault-tolerant routing on recursive dual-net," International Journal of Foundations of Computer Science, vol.22, no.5, pp.1001–1018, Aug. 2011.
[22] Y. Suzuki and K. Kaneko, "The container problem in bubble-sort graphs," IEICE Trans. Inf. & Syst., vol.E91-D, no.4, pp.1003–1009, April 2008.
[23] A. Bossard, "A set-to-set disjoint paths routing algorithm in hyper-star graphs," ISCA International Journal of Computers and Their Applications, vol.21, no.1, pp.76–82, March 2014.
[24] A. Bossard and K. Kaneko, "The set-to-set disjoint-path problem in perfect hierarchical hypercubes," The Computer Journal, vol.55, no.6, pp.769–775, June 2012.
[25] A. Bossard and K. Kaneko, "Time optimal node-to-set disjoint paths routing in hypercubes," Journal of Information Science and Engineering, vol.30, no.4, pp.1087–1093, July 2014.
[26] X.-B. Chen, "Many-to-many disjoint paths in faulty hypercubes," Information Sciences, vol.179, no.18, pp.3110–3115, Aug. 2009.
[27] Q.P. Gu, S. Okawa, and S. Peng, "Set-to-set fault tolerant routing in hypercudes," IEICE Trans. Fundamentals, vol.E79-A, no.4, pp.483–488, April 1996.
[28] 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, April 1996.
[29] X. Wang, J. Fan, S. Zhang, and J. Yu, "Node-to-set disjoint paths problem in cross-cubes," Journal of Supercomputing, vol.78, no.1, pp.1356–1380, Jan. 2022.
[30] K. Menger, "Zur allgemeinen Kurventhoerie," Fundamenta Mathematicae, vol.10, pp.96–115, 1927.

[31] L.R. Ford, Jr. and D.R. Fulkerson, "Maximal flow through a network," Canadian Journal of Mathematics, vol.8, pp.399–404, 1956.



Rikuya Sasaki is a master program student of Tokyo University of Agriculture and Technology in Japan. His main research areas are interconnection networks and fault-tolerant systems based on graph theory and network theory. He received the B.E. degree from Tokyo University of Agriculture and Technology in 2023.



Hiroyuki Ichida is a Ph.D. program student of Department of Electronic and Information Engineering, Graduate School of Engineering, Tokyo University of Agriculture and Technology in Japan. His research interests include dependable systems and graph theory. He received the B.E. and M.E. degrees from Tokyo University of Agriculture and Technology in 2019 and 2021, respectively.

Htoo Htoo Sandi Kyaw is an Assistant Professor at Tokyo University of Agriculture and Technology in Japan. Her main research areas are educational technology, web application systems, and graph theory. She received the B.E. and M.E. degrees from University of Technology (Yatanarpon Cyber City) in Myanmar in 2015 and 2018, respectively, and the Ph.D. degree from Okayama University in Japan in 2021.

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, 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.