Received 17 June 2022, accepted 30 June 2022, date of publication 6 July 2022, date of current version 15 July 2022.

Digital Object Identifier 10.1109/ACCESS.2022.3188783

RESEARCH ARTICLE

Set-to-Set Disjoint Path Routing in Bijective Connection Graphs

KEIICHI KANEKO[1], (Member, IEEE), ANTOINE BOSSARD[2], AND FREDERICK C. HARRIS, JR.[3]
[1]Institute of Engineering, Tokyo University of Agriculture and Technology, Tokyo 184-8588, Japan
[2]Graduate School of Science, Kanagawa University, Hiratsuka, Kanagawa 259-1293, Japan
[3]Department of Computer Science and Engineering, University of Nevada, Reno, Reno, NV 89557, USA

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 20K11729 and Grant 19K11887.

ABSTRACT The bijective connection graph encompasses a family of cube-based topologies, and n-dimensional bijective connection graphs include the hypercube and almost all of its variants with the order 2^n and the degree n. Hence, it is important to design and implement algorithms that work in bijective connection graphs. The set-to-set disjoint paths problem is as follows: given a set of source nodes $S = \{\boldsymbol{s}_1, \boldsymbol{s}_2, \ldots, \boldsymbol{s}_p\}$ and a set of destination nodes $D = \{\boldsymbol{d}_1, \boldsymbol{d}_2, \ldots, \boldsymbol{d}_p\}$ in a k-connected graph $G = (V, E)$ with $p \leq k$, construct p paths P_i: $\boldsymbol{s}_i \rightsquigarrow \boldsymbol{d}_{j_i}$ $(1 \leq i \leq p)$ such that $\{j_1, j_2, \ldots, j_p\} = \{1, 2, \ldots, p\}$ and the paths P_i are node-disjoint. Finding a solution to this problem is an important issue in parallel and distributed computation as well as the node-to-node disjoint paths problem and the node-to-set disjoint paths problem. In this paper we propose an algorithm that constructs p $(\leq n)$ disjoint paths between any pair of node sets in n-dimensional bijective connection graphs in polynomial-order time of n. We give a proof of correctness of the algorithm as well as the estimates of the time complexity $O(n^3p^4)$ and the maximum path length $n + p - 1$. According to a computer experiment in a locally twisted cube as an example of a bijective connection graph to construct n disjoint paths, the average time complexity of the algorithm is $O(n^2)$, and the average maximum path is $0.6333n - 0.266$.

INDEX TERMS Dependable computing, interconnection network, hypercube, multicomputer, parallel processing, performance evaluation supercomputers.

I. INTRODUCTION

For decades, research on parallel processing, especially on massively parallel systems, has been active. Because a massively parallel system connects many processing nodes, it is important to interconnect them efficiently. Therefore, many new topologies for interconnection networks have been proposed and studied instead of simple interconnection networks such as the ring, the mesh, the torus and the hypercube [1]. The bijective connection graph [2] provides a family of cube-based topologies. It is important to design and implement algorithms that work in a bijective connection graph because n-dimensional bijective connection graphs include the hypercube and almost all of its variants with the order 2^n and the degree n, including the twisted cube [3], the crossed cube [4], the Möbius cube [5], the locally twisted cube [6], the spined cube [7], and the twisted crossed cube [8]. Hence, such algorithms are enthusiastically studied [9]–[13].

The unsolved problems in the bijective connection graph include the set-to-set disjoint paths problem: given a set of source nodes $S = \{\boldsymbol{s}_1, \boldsymbol{s}_2, \ldots, \boldsymbol{s}_k\}$ and a set of destination nodes $D = \{\boldsymbol{d}_1, \boldsymbol{d}_2, \ldots, \boldsymbol{d}_k\}$ in a k-connected graph $G = (V, E)$, construct k paths P_i: $\boldsymbol{s}_i \rightsquigarrow \boldsymbol{d}_{j_i}$ $(1 \leq i \leq n)$ such that $\{j_1, j_2, \ldots, j_k\} = \{1, 2, \ldots, k\}$ and the paths P_i are node-disjoint. The set-to-set disjoint paths problem is an important issue in parallel and distributed computation [14]–[22] along with the node-to-node disjoint paths problem [23]–[35] and the node-to-set disjoint paths problem [36]–[45]. Finding disjoint paths in a massively parallel system has many applications. For instance, multiple pairs of nodes can establish

The associate editor coordinating the review of this manuscript and approving it for publication was Thomas Canhao Xu.

full-bandwidth high-speed communication over a network by circuit switching. Circuit switching attains an optimal data transfer performance because it does not conduct any switching inside the routers of intermediate nodes. Also, circuit switching does not allow any interference with other communications, thereby ensuring security and privacy. Another advantage is that disjoint paths routing prevents deadlocks, livelocks, and starvations, and does not require any recovery process, making it suitable for real-time applications. Furthermore, disjoint paths routing significantly increases the system dependability for the following reason: If multiple paths have a common node, that node is crucial to their dependability. However, one faulty node can block at most one constructed path in the disjoint paths routing. Hence, the disjoint paths routing drastically improves the robustness of a massively parallel system. If the set-to-set disjoint paths problem is solved, its approach can be applied to solve the node-to-node and node-to-set disjoint paths problems. Hence, the set-to-set disjoint paths problem is hardest but most important.

For a graph $G = (V, E)$, the set-to-set disjoint paths can be constructed in polynomial-order time of $|V|$, using the maximum flow algorithm. However, the complexity of the algorithm is too large for an n-dimensional bijective connection graph (B_n) because the number of nodes in it is equal to 2^n. In this paper we propose an algorithm called S2S (set-to-set) that solves the problem in cube-based topologies in a polynomial-order time of n instead of 2^n. We also present the results of an average performance evaluation by a computer experiment. Algorithm S2S consists of three cases according to the relative positions of the source nodes and the destination nodes. The third case is divided into three sub-cases. The algorithm constructs $p(\leq n)$ disjoint paths between the set of source nodes S and the set of destination nodes D where $|S| = |D| = p$ and n is equal to the connectivity of a B_n. Because cube-based topologies are popular for interconnection networks, the generalized disjoint paths routing method provided by our algorithm is useful as there is no need to design a specific algorithm for each of the topologies that belong to the bijective connection graph.

The remainder of this paper is organized as follows: Section 2 describes the related works regarding the disjoint paths problems. Section 3 introduces the definition of bijective connection graphs as well as other requisite definitions and a routing algorithm R, which can tolerate the existence of a single faulty node in a bijective connection graph. Section 4 introduces a fault-tolerant routing algorithm FTR, which will be used in S2S. Section 5 explains Algorithm S2S in detail. Section 6 describes a proof of correctness and the theoretical complexities of S2S. The average performance of S2S is reported in Section 7. We conclude and give future work in Section 8.

II. RELATED WORK

In this section we introduce the related works on three disjoint paths problems: node-to-node, node-to-set, and set-to-set in this order regarding the typical topologies proposed for interconnection networks. The studies for the cube-based topologies are summarized in Table 1.

There are many studies regarding the node-to-node disjoint paths problem. The algorithm in [27] solves the problem in n-burnt pancake graphs in $O(n^3)$ time, with a maximum length of constructed paths of $3n + 4$. The algorithm in [34] solves the problem in an n-bubble-sort graph in $O(n^4)$ time, with a maximum length of constructed paths of $n(n + 1)/2$. The algorithm in [23] solves the problem in a hierarchical dual-net in $O((d_0 + k)2^k)$ time, where d_0 is the node degree of the symmetric base network B, and k is the level, that is, the number of recursive construction. The maximum length of constructed paths is $(3 \cdot 2^{k-1} + 2)diam(B) - 3\sum_{j=0}^{k-2} 2^j diam(SN^{k-1-j}) - diam(SN^k) + 3 \cdot 2^k + 2k - 2$, where SN^i is a super-node of level i. The algorithm in [25] solves the problem in an n-dimensional hierarchical cubic network, with a maximum length of constructed paths of $n + \lfloor n/3 \rfloor + 4$. The algorithm in [30] solves the problem in a recursive dual-net in $O((d_0+k)2^k t)$ time, where d_0 is the node degree of the symmetric base network B, k is the level, and $O(t)$ is the time complexity to construct a path in B. The maximum length of constructed paths is $2^k diam(B) + 2^{k+1} - 2$. The algorithm in [35] solves the problem in a hierarchical hypercube network with degree $n+1$, with a maximum length of constructed paths of $\max\{2^{n+1} + 2n + 1, 2^{n+1} + n + 4\}$. The algorithm in [29] solves the problem in an (n, k)-star graph, $S_{n,k}$, with $2 \leq k \leq n - 2$, with a maximum length of constructed paths of $diam(S_{n,k}) + 2$ for $2 \leq k \leq \lfloor n/2 \rfloor$ and $diam(S_{n,k}) + 1$ or $diam(S_{n,k}) + 2$ for $\lfloor n/2 \rfloor + 1 \leq k \leq n - 2$. It is proved in [32] that there are c $(1 \leq c \leq 2n)$ disjoint paths between two nodes in a k-ary n-dimensional torus such that they contain all of the nodes in the torus. The torus with an even k is bipartite. Hence, the two nodes must be from distinct partite sets.

There are also many studies regarding the node-to-set disjoint paths problem. The algorithm in [36] solves the problem in a (k, n)-torus connected cycles network in $O(n^3+kn^2)$ time, with a maximum length of constructed paths of $\lfloor k/2 \rfloor n^2 + (\lfloor k/2 \rfloor + 5)n - 4$. The algorithm in [37] solves the problem in an n-star graph in $O(n^2)$ time, with a maximum length of constructed paths of $\lfloor 3(n - 1)/2 \rfloor + 2$. The algorithm in [41] solves the problem in an n-dimensional folded hypercube in $O(n^3)$ time, with a maximum length of constructed paths of $\lceil n/2 \rceil + 1$. The algorithm in [42] solves the problem in an n-dimensional folded hypercube in $O(n^3)$ time, with a maximum length of constructed paths of the maximum distance between the source node and the $(n + 1)$ destination nodes plus 2. The algorithm in [43] solves the problem in a $(2n, n)$-hyper-star network in $O(n^5)$ time, with a maximum length of constructed paths of the maximum distance between the source node and the destination nodes plus 4. The algorithm in [44] solves the problem in a recursive dual-net in $O(((d_0 + k)diam(B)/\log n_0) \log N)$ time, where d_0 is the node degree of the base network B, k is the level, n_0 is the number of nodes in B, and N is the number of the nodes in the recursive dual-net $(N = (2n_0)^{2^k}/2)$. The maximum length of constructed

TABLE 1. **Comparison of *n*-dimensional cube-based topologies regarding node-disjoint path routing algorithms in constructing *n* disjoint paths.**

topology	diameter	node-to-node		node-to-set		set-to-set	
		time	length	time	length	time	length
Hypercube	n	$O(n^2)$ [46]	$n+1$ [46]	$O(n^2)$ [47]	$n+1$ [47]	$O(n^2 \log n)$ [20]	$2n+2$ [20]
Twisted Cube	$\lfloor (n+1)/2 \rfloor$	—	$\lceil n/2 \rceil + 2^{\S}$ [48]	—	—	—	—
Crossed Cube	$\lfloor (n+1)/2 \rfloor$	$O(n^2)$ [49]	$3n-5$ [49]	—	—	—	—
0-Möbius Cube	$\lceil (n+2)/2 \rceil$	$O(n^2)$ [28]	$3n-5$ [28]	$O(n^4)$ [40]	$2n-1$ [40]	$O(n^6)$ [50]	$4n-7$ [50]
1-Möbius Cube	$\lceil (n+1)/2 \rceil$	$O(n^2)$ [28]	$3n-5$ [28]	$O(n^4)$ [40]	$2n-1$ [40]	$O(n^6)$ [50]	$4n-7$ [50]
Locally Twisted Cube	$\lceil (n+3)/2 \rceil^{\dagger}$	—	—	—	—	—	—
Spined Cube	$\lceil n/3 \rceil + 3^{\ddagger}$	—	—	—	—	—	—
Twisted Crossed Cube	$\lceil (n+1)/2 \rceil$	$O(n^2)$ [51]	$4n-8$ [51]	—	—	—	—

$\dagger$: $n \geq 5$, $\ddagger$: $n \geq 14$, $\S$: n is odd.

paths is $3(diam(B)/2 + 1)(\log_2 N + 1)/(\log_2 n_0 + 1)$. The algorithm in [45] solves the problem in an (n, k)-star graph in $O(k^2 n^2)$ time, with a maximum length of constructed paths of $6k - 7$.

There are several studies on the set-to-set disjoint paths problem. The algorithm in [21] solves the problem in a k-ary n-dimensional torus in $O(kn^3 + n^3 \log n)$ time, with a maximum length of constructed paths of $2(k+1)n$. The algorithm in [15] solves the problem in a $(2^n + n)$-dimensional perfect hierarchical hypercube in $O(n^2 2^{2n})$ time, with a maximum length of constructed paths of $(n+1)(2^n + n + 4) + 3$. The algorithm in [16] solves the problem in an n-dimensional hierarchical cubic network in $O(n^2 \log n)$ time, with a maximum length of constructed paths of $6n + 3$. The algorithm in [22] solves the problem in an n-dimensional hypercube with f faulty nodes. The algorithm can construct k disjoint paths between two node sets that include at least $(2^n - 2f)$ nodes if $f \leq 2n - 2f - 2$ and each non-faulty node has at least two non-faulty neighbor nodes. The algorithm in [17] solves the problem in an n-dimensional hypercube with f_{n} faulty nodes and f_{e} faulty edges. The algorithm can construct p disjoint paths between two node sets that include at least $(2^n - 2f_v\mathrm{n})$ nodes if $f_{\mathrm{n}} + f_{\mathrm{e}} \leq n - p - 1$. The algorithm in [18] solves the problem in an n-dimensional hypercube with at most $(n-p)$ faulty nodes to construct p disjoint paths between two node sets in $O(pn \log p)$ time. The maximum length of constructed paths is $n + p$.

As mentioned above, there have been many studies on the disjoint paths problems, producing many important results. However, there are still many topologies for which these problems are unsolved. Table 1 shows a restricted list of typical cube-based topologies, indicating the disjoint paths problems that have not yet been solved.

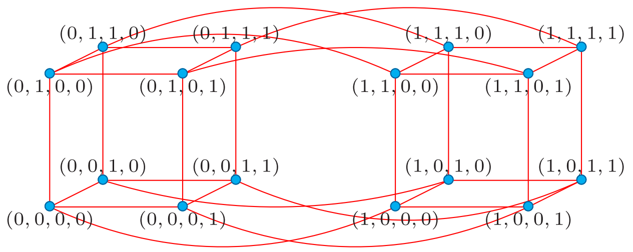

FIGURE 1. **Example of a 4-dimensional hypercube, which belongs to $\mathcal{L}_4$.**

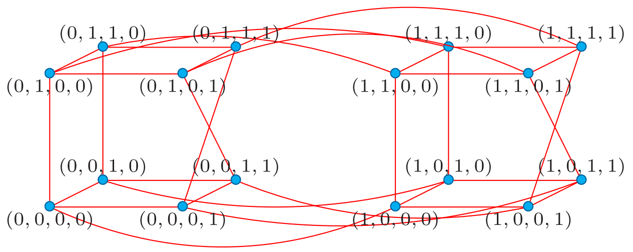

FIGURE 2. **Example of a 4-dimensional twisted cube, which belongs to $\mathcal{L}_4$.**

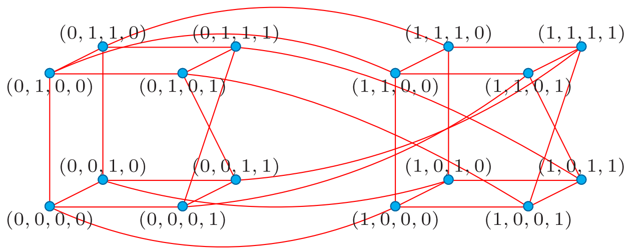

FIGURE 3. **Example of a 4-dimensional locally twisted cube, which belongs to $\mathcal{L}_4$.**

III. PRELIMINARIES

In this section we first give a definition of bijective connection (BC) graphs.

Definition 1: A class of n-dimensional BC graphs $\mathcal{L}_n$ is recursively defined as follows: $\mathcal{L}_0 = \{B_0\}$ where $|V(B_0)| = 1$ and $E(B_0) = \emptyset$. Note that B_0 consists of a single node. $B_n \in \mathcal{L}_n$ if and only if there are two node-disjoint BC graphs B^0_{n-1} and B^1_{n-1} in $\mathcal{L}_{n-1}$ such that $V(B_n) = V(B^0_{n-1}) \cup V(B^1_{n-1})$, and $E(B_n) = E(B^0_{n-1}) \cup E(B^1_{n-1}) \cup E_n$ where $E_n = \{(\boldsymbol{x}, \phi_n(\boldsymbol{x})) \mid \boldsymbol{x} \in V(B^0_{n-1})\}$ with a bijection $\phi_n : V(B^0_{n-1}) \to V(B^1_{n-1})$.

Figures 1, 2, and 3 show the 4-dimensional hypercube, the 4-dimensional twisted cube, and the 4-dimensional locally twisted cube, respectively. They all belong to $\mathcal{L}_4$. Figure 4 shows the cube-connected cycles [52], which do not belong to BC graphs.

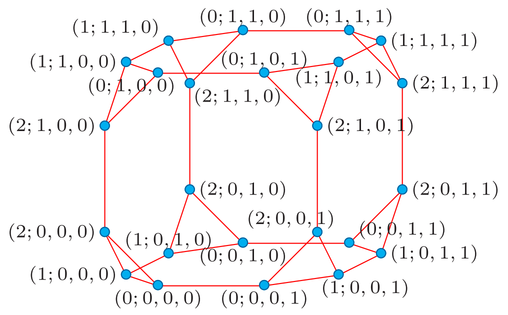

FIGURE 4. Example of cube-connected cycles, which do not belong to BC graphs.

Because we evaluate our algorithm based on the locally twisted cube in a later section, we formally define it here.

Definition 2: An n-dimensional locally twisted cube is an undirected graph whose node set is $\{0, 1\}^n$. *For each node* $\boldsymbol{a} = (a_1, a_2, \ldots, a_n)$, *it has n adjacent nodes:* $(a_1, a_2, \ldots, a_{n-1}, \overline{a_n})$, $(a_1, a_2, \ldots, a_{n-2}, \overline{a_{n-1}}, a_n)$, *and* $(a_1, a_2, \ldots, a_{i-1}, \overline{a_i}, a_{i+1} \oplus a_n, a_{i+2}, \ldots, a_n)$ $(1 \leq i \leq n-2)$ *where* $\overline{a}$ $(a \in \{0, 1\})$ *represents* $1 - a$ *and* $\oplus$ *represents the bitwise exclusive-or operation with* $0 \oplus 0 = 1 \oplus 1 = 0$ *and* $0 \oplus 1 = 1 \oplus 0 = 1$.

In the rest of this paper, for a node $\boldsymbol{x} \in V(B_n)$, we use $\boldsymbol{x}^{(j)}$ and $\boldsymbol{x}^{(j_1, j_2)}$ to represent $\phi_j(\boldsymbol{x})$ and $\phi_{j_2}(\phi_{j_1}(\boldsymbol{x}))$, respectively. In addition, we assume that B_n consists of B^0_{n-1} and B^1_{n-1}, and B^i_{n-1} $(i \in \{0, 1\})$ consists of B^{i0}_{n-2} and B^{i1}_{n-2}.

For two nodes $\boldsymbol{x}, \boldsymbol{y} \in V(B_n)$, we assume that we can check if $\boldsymbol{x} = \boldsymbol{y}$ in $O(1)$ time. We also assume that we can check if $\boldsymbol{x} \in B^0_{n-1}$ or $\boldsymbol{x} \in B^1_{n-1}$ in $O(1)$ time.

In this paper a path is an alternating finite sequence of nodes and edges, $\boldsymbol{a}_0, e_0, \boldsymbol{a}_1, e_1, \boldsymbol{a}_2, \ldots, \boldsymbol{a}_{k-1}, e_{k-1}, \boldsymbol{a}_k$ where $e_i = (\boldsymbol{a}_i, \boldsymbol{a}_{i+1})$ $(0 \leq i \leq k-1)$. The length of a path is the number of edges included in it. We use the notation $\boldsymbol{a}_0 \rightarrow \boldsymbol{a}_1 \rightarrow \cdots \rightarrow \boldsymbol{a}_k$ or $\boldsymbol{a}_0 \rightsquigarrow \boldsymbol{a}_k$ as long as it does not cause confusion. Two paths are node-disjoint if they do not have any common node. For simplicity, we sometimes use 'disjoint' instead of 'node-disjoint' in the rest of this paper.

Lemma 1: There is no cycle whose length is 3 in a BC graph.

Proof: B_n $(n \leq 2)$ does not contain a cycle whose length is 3. Assume that there is no cycle whose length is 3 in B_{k-1}. Then, if there is a cycle that consists of three nodes $\boldsymbol{x}$, $\boldsymbol{y}$, and $\boldsymbol{z}$ in B_k, we can assume without loss of generality that $\boldsymbol{x}, \boldsymbol{y} \in V(B^i_{k-1})$ and $\boldsymbol{z} \in V(B^{\bar{i}}_{k-1})$ for $i(\in \{0, 1\})$, and $(\boldsymbol{x}, \boldsymbol{y}), (\boldsymbol{x}, \boldsymbol{z}), (\boldsymbol{y}, \boldsymbol{z}) \in E(B_k)$. However, from Definition 1, the nodes in B^i_{k-1} and $B^{\bar{i}}_{k-1}$ are connected by a bijection. Hence, $\boldsymbol{x} = \boldsymbol{y}$, and it is a contradiction. Therefore, there is no cycle of length 3 in B_k. By mathematical induction, we can conclude that there is no cycle of length 3 in B_n $(n \geq 0)$. □

Lemma 2: For B_n, *which consists of two node-disjoint BC graphs* B^0_{n-1} *and* B^1_{n-1}, *assume a node* $\boldsymbol{a}$ *is in* B^i_{n-1} $(i \in \{0, 1\})$. *Then, we can construct n paths from* $\boldsymbol{a}$ *to* $B^{\bar{i}}_{n-1}$: P_j: $\boldsymbol{a}(\in V(B^i_{n-1})) \rightsquigarrow \boldsymbol{b}_j(\in V(B^{\bar{i}}_{n-1}))$ $(1 \leq j \leq n)$ *in* $O(n)$ *time*

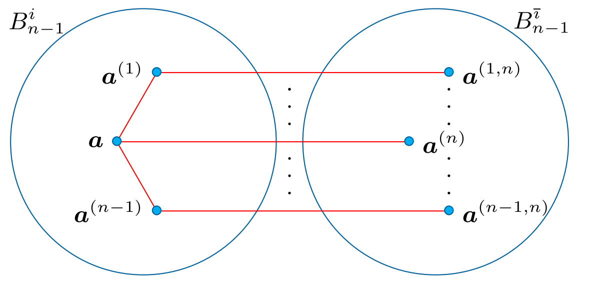

FIGURE 5. ***n*** **disjoint paths from node** $\boldsymbol{a}(\in B^i_{n-1})$ **to** $B^{\bar{i}}_{n-1}$**.**

such that the paths are disjoint except for $\boldsymbol{a}$ *and their lengths are at most* 2.

Proof: For $\boldsymbol{a}$, we can construct n disjoint paths:

$$P_j : \begin{cases} \boldsymbol{a} \rightarrow \boldsymbol{a}^{(j)} \rightarrow \boldsymbol{a}^{(j,n)} & (1 \leq j \leq n-1), \\ \boldsymbol{a} \rightarrow \boldsymbol{a}^{(n)} & (j = n). \end{cases}$$

In Fig. 5 the length of the path P_n is 1, and the lengths of the remaining $(n-1)$ paths P_j $(1 \leq j \leq n-1)$ are all 2. It takes $O(1)$ time to construct a path. Hence, it takes $O(n)$ time in total. □

Lemma 3: For two non-faulty nodes $\boldsymbol{s}, \boldsymbol{d}(\in V(B_n))$ *and at most one faulty node* $\boldsymbol{f}(\in V(B_n))$, *there is an algorithm, R, that constructs a fault-free path* $\boldsymbol{s} \rightsquigarrow \boldsymbol{d}$ *of length at most n in* $O(n)$ *time.*

Proof: If $\boldsymbol{s} = \boldsymbol{d}$, R can construct the path of length 0. Hence, we assume that $\boldsymbol{s} \neq \boldsymbol{d}$. We also assume that $\boldsymbol{s} \in B^i_{n-1}$ $(i \in \{0, 1\})$. Then, R is given as follows.

Step 1) If $n = 1$, then since $\boldsymbol{s}$ and $\boldsymbol{d}$ are adjacent in B_1, select the edge $\boldsymbol{s} \rightarrow \boldsymbol{d}$, and terminate.

Step 2) If $\boldsymbol{d} \in V(B^i_{n-1})$, then apply R in B^i_{n-1} recursively to construct a fault-free path between $\boldsymbol{s}$ and $\boldsymbol{d}$, and terminate.

Step 3) If $\boldsymbol{f} \in V(B^i_{n-1})$, select the edge $\boldsymbol{s} \rightarrow \boldsymbol{s}^{(n)}$, and apply R in $B^{\bar{i}}_{n-1}$ recursively to construct a fault-free path between $\boldsymbol{s}^{(n)}$ and $\boldsymbol{d}$. Otherwise, select the edge $\boldsymbol{d} \rightarrow \boldsymbol{d}^{(n)}$, and apply R in B^i_{n-1} recursively to construct a fault-free path between $\boldsymbol{s}$ and $\boldsymbol{d}^{(n)}$.

Since each step takes only $O(1)$ time to select at most one edge and the steps are executed at most n times, Algorithm R constructs a fault-free path from $\boldsymbol{s}$ to $\boldsymbol{d}$ of length at most n in $O(n)$ time. □

IV. FAULT-TOLERANT ROUTING

For a source node $\boldsymbol{s}$ and a destination node $\boldsymbol{d}$ in an n-dimensional BC graph with a set of faulty nodes F such that $|F| = f \leq n-1$, the following algorithm, FTR, gives a fault-free path. If $f \leq 1$, we can apply R to construct a fault-free path between $\boldsymbol{s}$ and $\boldsymbol{d}$, hence, we assume that $2 \leq f < n$. The algorithm is divided into two cases depending on the distribution of the source and destination nodes.

A. CASE 1

Assume that $\boldsymbol{s}, \boldsymbol{d} \in V(B^i_{n-1})$ $(i \in \{0, 1\})$.

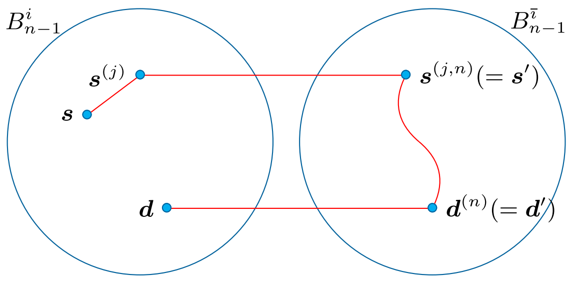

FIGURE 6. After Step 4 in Case 1 of FTR.

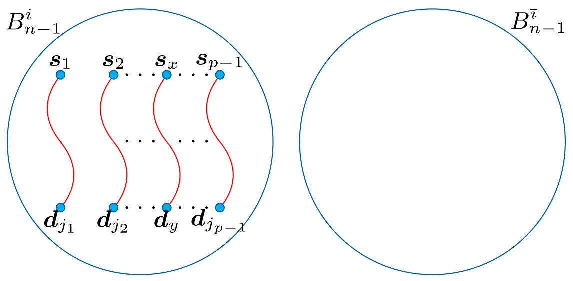

FIGURE 8. After Step 2 in Case 1 of S2S.

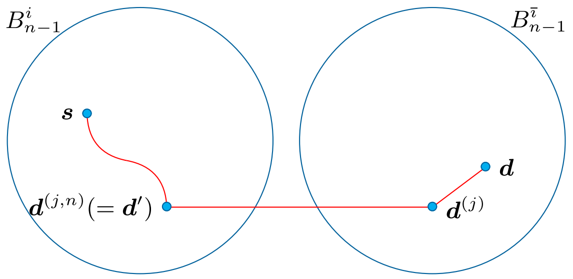

FIGURE 7. After Step 2 in Case 2 of FTR.

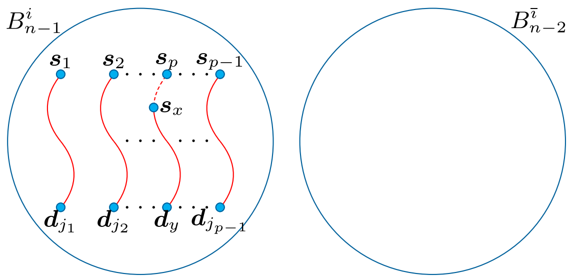

FIGURE 9. After Step 3 in Case 1 of S2S.

Step 1) If $|V(B^i_{n-1}) \cap F| \leq |V(B^{\bar{i}}_{n-1}) \cap F|$, apply FTR in B^i_{n-1} recursively to construct a fault-free path between $\boldsymbol{s}$ and $\boldsymbol{d}$, and terminate.

Step 2) If $\boldsymbol{s}^{(n)} \notin F$, select an edge $\boldsymbol{s} \to \boldsymbol{s}^{(n)}(=\boldsymbol{s}')$. Otherwise, select a fault-free path $\boldsymbol{s} \to \boldsymbol{s}^{(j)} \to \boldsymbol{s}^{(j,n)}(=\boldsymbol{s}')$ $(1 \leq j \leq n-1)$.

Step 3) If $\boldsymbol{d}^{(n)} \notin F$, select an edge $\boldsymbol{d} \to \boldsymbol{d}^{(n)}(=\boldsymbol{d}')$. Otherwise, select a fault-free path $\boldsymbol{d} \to \boldsymbol{d}^{(j)} \to \boldsymbol{d}^{(j,n)}(=\boldsymbol{d}')$ $(1 \leq j \leq n-1)$.

Step 4) Apply FTR in $B^{\bar{i}}_{n-1}$ recursively to construct a fault-free path between $\boldsymbol{s}'$ and $\boldsymbol{d}'$. See Fig. 6.

B. CASE 2

Assume that $\boldsymbol{s} \in V(B^i_{n-1})$ and $\boldsymbol{d} \in V(B^{\bar{i}}_{n-1})$ $(i \in \{0, 1\})$. Without loss of generality, we can assume that $|V(B^i_{n-1}) \cap F| \leq |V(B^{\bar{i}}_{n-1}) \cap F|$.

Step 1) If $\boldsymbol{d}^{(n)} \notin F$, select an edge $\boldsymbol{d} \to \boldsymbol{d}^{(n)}(=\boldsymbol{d}')$. Otherwise, select a fault-free path $\boldsymbol{d} \to \boldsymbol{d}^{(j)} \to \boldsymbol{d}^{(j,n)}(=\boldsymbol{d}')$ $(1 \leq j \leq n-1)$.

Step 2) Apply FTR in B^i_{n-1} recursively to construct a fault-free path between $\boldsymbol{s}$ and $\boldsymbol{d}'$. See Fig. 7.

V. SET-TO-SET DISJOINT PATH ROUTING

For a set of p source nodes $S = \{\boldsymbol{s}_1, \boldsymbol{s}_2, \ldots, \boldsymbol{s}_p\}$ and a set of p destination nodes $D = \{\boldsymbol{d}_1, \boldsymbol{d}_2, \ldots, \boldsymbol{d}_p\}$ in an n-dimensional BC graph $(p \leq n)$, the following algorithm, S2S, gives p disjoint paths between S and D. If $n = 1$ or $p = 1$, we can use Algorithm R to construct a path between $\boldsymbol{s}_1$ and $\boldsymbol{d}_1$ of length at most n in $O(n)$ time; hence, we assume that $2 \leq p \leq n$. The algorithm is divided into three cases depending on the distribution of the source nodes and the destination nodes.

A. CASE 1

Assume that $(S \cup D) \subset V(B^i_{n-1})$ for $i(\in \{0, 1\})$.

Step 1) If $p < n$, apply S2S recursively in B^i_{n-1} to construct p disjoint paths, and terminate.

Step 2) Apply S2S recursively in B^i_{n-1} to construct $(p-1)$ disjoint paths between $\{\boldsymbol{s}_1, \boldsymbol{s}_2, \ldots, \boldsymbol{s}_{p-1}\}$ and $\{\boldsymbol{d}_1, \boldsymbol{d}_2, \ldots, \boldsymbol{d}_{p-1}\}$. See Fig. 8.

Step 3) If $\boldsymbol{s}_p$ is included in one of the paths constructed in Step 2, say $\boldsymbol{s}_x \rightsquigarrow \boldsymbol{d}_y$, discard the sub path $\boldsymbol{s}_x \rightsquigarrow \boldsymbol{s}_p$, and exchange the indices of $\boldsymbol{s}_x$ and $\boldsymbol{s}_p$. See Fig. 9.

Step 4) If $\boldsymbol{d}_p$ is included in one of the paths constructed in Steps 2 and 3, say $\boldsymbol{s}_x \rightsquigarrow \boldsymbol{d}_y$, discard the sub path $\boldsymbol{d}_p \rightsquigarrow \boldsymbol{d}_y$, and exchange the indices of $\boldsymbol{d}_y$ and $\boldsymbol{d}_p$.

Step 5) If $\boldsymbol{s}_p = \boldsymbol{d}_p$, terminate. Otherwise, select the edges $\boldsymbol{s}_p \to \boldsymbol{s}_p^{(n)}(=\boldsymbol{s}'_p)$ and $\boldsymbol{d}_p \to \boldsymbol{d}_p^{(n)}(=\boldsymbol{d}'_p)$.

Step 6) Apply R in $B^{\bar{i}}_{n-1}$ to construct a path between $\boldsymbol{s}'_p$ and $\boldsymbol{d}'_p$.

B. CASE 2

Assume that $0 < |S \cap V(B^i_{n-1})|$, $0 < |D \cap V(B^i_{n-1})|$, and $0 < |(S \cup D) \cap V(B^{\bar{i}}_{n-1})|$ for $i(\in \{0, 1\})$. Without loss of generality, we can assume that $S \cap V(B^i_{n-1}) = \{\boldsymbol{s}_1, \boldsymbol{s}_2, \ldots, \boldsymbol{s}_l\}$, $D \cap V(B^i_{n-1}) = \{\boldsymbol{d}_1, \boldsymbol{d}_2, \ldots, \boldsymbol{d}_m\}$, and $l \leq m \leq p$.

Step 1) Apply S2S recursively in B^i_{n-1} to construct l disjoint paths between $\{\boldsymbol{s}_1, \boldsymbol{s}_2, \ldots, \boldsymbol{s}_l\}$ and $\{\boldsymbol{d}_1, \boldsymbol{d}_2, \ldots, \boldsymbol{d}_l\}$.

Step 2) For each of $\boldsymbol{d}_k$ $(l+1 \leq k \leq m)$, if $\boldsymbol{d}_k$ is included in one of the paths constructed in Step 1, say $\boldsymbol{s}_x \rightsquigarrow \boldsymbol{d}_y$, discard the sub path $\boldsymbol{d}_k \rightsquigarrow \boldsymbol{d}_y$, and exchange the indices of $\boldsymbol{d}_y$ and $\boldsymbol{d}_k$.

Step 3) For each of $\boldsymbol{d}_k$ $(l+1 \leq k \leq m)$, if $\boldsymbol{d}_k^{(n)} \notin D$, select the edge $\boldsymbol{d}_k \to \boldsymbol{d}_k^{(n)}(=\boldsymbol{d}'_k)$. Otherwise, if there is a path

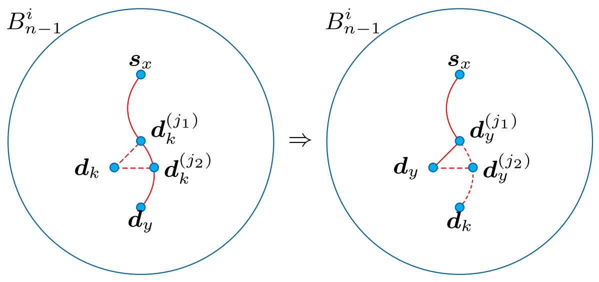

FIGURE 10. Exchanging d_y and d_k in Step 3 in Case 2 of S2S.

$d_k \to d^{(j)}_k \to d^{(j,n)}_k$ $(1 \le j \le n-1)$ that is disjoint from other destination nodes and the paths from them, select it and let d'_k be $d^{(j,n)}_k$. Otherwise, there is a path, say $s_x \rightsquigarrow d_y$ constructed in Steps 1 and 2 that includes at least two neighbor nodes $d^{(j_1)}_k$ and $d^{(j_2)}_k$ of the node d_k. Without loss of generality, we can assume that $d^{(j_1)}_k$ is located closer to s_x on the path than $d^{(j_2)}_k$, that is, $s_x \rightsquigarrow d^{(j_1)}_k \rightsquigarrow d^{(j_2)}_k \rightsquigarrow d_y$. Then, discard the sub path $d^{(j_1)}_k \rightsquigarrow d^{(j_2)}_k \rightsquigarrow d_y$, add the edge $d^{(j_1)}_k \to d_k$ to construct the path $s_x \rightsquigarrow d^{(j_1)}_k \to d_k$, exchange the indices of d_y and d_k, and try to find a disjoint path from d_k again. See Fig. 10.

Step 4) Apply S2S recursively in $B^{\bar{i}}_{n-1}$ to construct $(p-l)$ disjoint paths between $\{s_{l+1}, s_{l+2}, \dots, s_p\}$ and $\{d'_{l+1}, d'_{l+2}, \dots, d'_m, d_{m+1}, d_{m+2}, \dots, d_p\}$.

C. CASE 3

Assume that $S \subset V(B^{i_1}_{n-1})$ and $D \subset V(B^{\bar{i}_1}_{n-1})$ for $i_1 (\in \{0,1\})$. If $p < n$, we can select the edges $d_k \to d^{(n)}_k$ $(1 \le k \le p)$ and apply S2S recursively to construct p disjoint paths in $B^{\bar{i}_1}_{n-1}$; hence, we assume that $p = n$.

1) CASE 3-1

Assume that $|S \cap V(B^{i_1 i_2}_{n-2})| \ge p-1$ for $i_2 (\in \{0,1\})$. Without loss of generality, we can assume that $\{s_1, s_2, \dots, s_{p-1}\} \subset V(B^{i_1 i_2}_{n-2})$.

Step 1) For each s_k $(1 \le k \le p-1)$, select the edge $s_k \to s^{(n)}_k (= s'_k)$.

Step 2) Apply S2S recursively in $B^{\bar{i}_1}_{n-1}$ to construct $(p-1)$ disjoint paths between $\{s'_1, s'_2, \dots, s'_{p-1}\}$ and $\{d_1, d_2, \dots, d_{p-1}\}$.

Step 3) If d_p is included in one of the paths constructed in Step 2, say $s'_x \rightsquigarrow d_y$, discard the sub path $d_p \rightsquigarrow d_y$, and exchange the indices of d_y and d_p.

Step 4) Select the edge $d_p \to d^{(n)}_p$. If $d^{(n)}_p = s_p$, terminate.

Step 5) If $s_p \in V(B^{i_1 i_2}_{n-2})$, select the edge $s_p \to s^{(n-1)}_p (= s'_p)$. Otherwise, let s'_p be s_p.

Step 6) If $d^{(n)}_p \in B^{i_1 i_2}_{n-2}$, select the edge $d^{(n)}_p \to d^{(n,n-1)}_p (= d'_p)$. Otherwise, let d'_p be $d^{(n)}_p$.

Step 7) Apply R in $B^{i_1 \bar{i}_2}_{n-2}$ to construct a path between s'_p and d'_p. See Fig. 11.

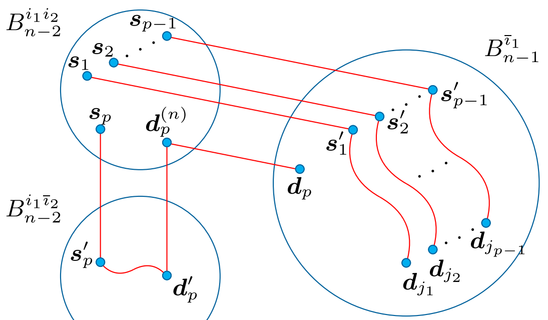

FIGURE 11. After Step 7 in Case 3-1 of S2S with $s_p, d^{(n)}_p \in B^{i_1 i_2}_{n-2}$.

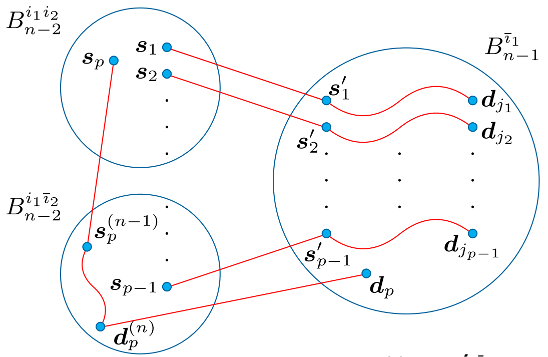

FIGURE 12. After Step 6 in Case 3-2 of S2S with $d^{(n)}_p \in V(B^{i_1 \bar{i}_2}_{n-2})$ and $s^{(n-1)}_p \notin S$.

2) CASE 3-2

Assume that $2 \le |S \cap V(B^{i_1 i_2}_{n-2})| \le p-2$ for $i_2 (\in \{0,1\})$. Without loss of generality, we can assume that $|S \cap V(B^{i_1 \bar{i}_2}_{n-2})| \le |S \cap V(B^{i_1 i_2}_{n-2})|$ and $s_p \in V(B^{i_1 i_2}_{n-2})$. If $n = p = 4$, $\lfloor n/2 \rfloor = n-2 = 2$. Hence, from $|S \cap V(B^{i_1 \bar{i}_2}_{n-2})| = |S \cap V(B^{i_1 i_2}_{n-2})| = 2$, two neighbor nodes of $s^{(n-1)}_p$ in $B^{i_1 \bar{i}_2}_{n-2}$ may be both source nodes, and it is not possible to construct a disjoint path from $s^{(n-1)}_p$ to $d^{(n)}$ by applying FTR in $B^{i_1 \bar{i}_2}_{n-2}$ in Step 6 of Case 3-2. Therefore, if $n = p = 4$, we apply Case 3-3.

Step 1) If there is a source node $s_x (\in V(B^{i_1 i_2}_{n-2}))$ such that $s^{(n-1)}_x \notin S$, exchange the indices of s_x and s_p.

Step 2) For each s_k $(1 \le k \le p-1)$, select the edge $s_k \to s^{(n)}_k (= s'_k)$.

Step 3) Apply S2S recursively in $B^{\bar{i}_1}_{n-1}$ to construct $(p-1)$ disjoint paths between $\{s'_1, s'_2, \dots, s'_{p-1}\}$ and $\{d_1, d_2, \dots, d_{p-1}\}$.

Step 4) If d_p is included in one of the paths constructed in Step 3, say $s'_x \rightsquigarrow d_y$, discard the sub path $d_p \rightsquigarrow d_y$, and exchange the indices of d_y and d_p.

Step 5) Select the edge $d_p \to d^{(n)}_p$. If $d^{(n)}_p = s_p$, terminate.

Step 6) If $d^{(n)}_p \in V(B^{i_1 \bar{i}_2}_{n-2})$ and $s^{(n-1)}_p \notin S$, select the edge $s_p \to s^{(n-1)}_p$, apply FTR in $B^{i_1 \bar{i}_2}_{n-2}$ to construct a fault-free path between $s^{(n-1)}_p$ and $d^{(n)}_p$ by regarding the nodes in $S \cap V(B^{i_1 \bar{i}_2}_{n-2})$ as faulty, and terminate. See Fig. 12.

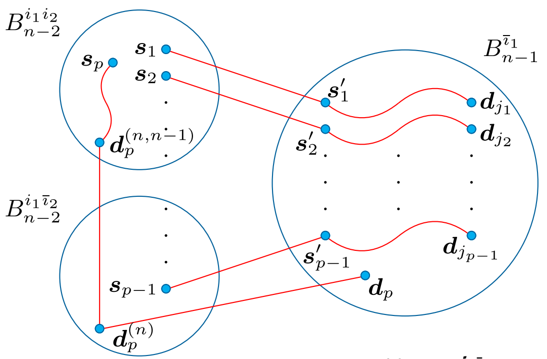

FIGURE 13. After Step 8 in Case 3-2 of S2S with $d_p^{(n)} \in V(B^{i_1\bar{i}_2}_{n-2})$ and $s_p^{(n-1)} \in S$.

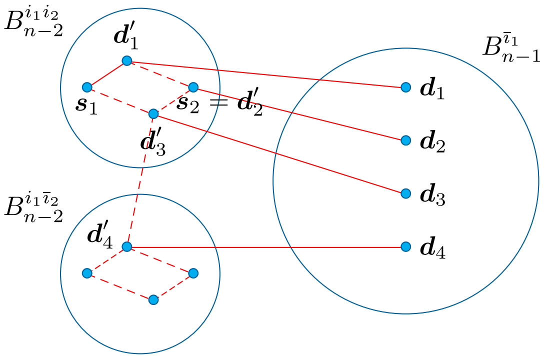

FIGURE 14. In Step 3 in Case 3-3 of S2S with $d'_3 \in V(B^{i_1 i_2}_{n-2})$ and $d'^{(n-1)}_3 = d'_4$.

Step 7) If $d_p^{(n)} \in V(B^{i_1\bar{i}_2}_{n-2})$, select the edge $d_p^{(n)} \to d_p^{(n,n-1)} (= d'_p)$. Otherwise, let d'_p be $d_p^{(n)}$.

Step 8) Apply FTR in $B^{\bar{i}_1}_{n-1}$ to construct a fault-free path between s_p and d'_p by regarding the nodes in $(S \setminus \{s_p\}) \cap V(B^{i_1 i_2}_{n-2})$ as faulty. See Fig. 13.

3) CASE 3-3

Assume that $n = p = 4$ and $|S \cap V(B^{i_1\bar{i}_2}_{n-2})| = |S \cap V(B^{i_1 i_2}_{n-2})| = 2$ for $i_2 (\in \{0, 1\})$. We can assume without loss of generality that $S \cap V(B^{i_1 i_2}_{n-2}) = \{s_1, s_2\}$ and $S \cap V(B^{i_1\bar{i}_2}_{n-2}) = \{s_3, s_4\}$.

Step 1) Select edges $d_k \to d_k^{(n)} (= d'_k)$ $(1 \le k \le n)$. Let $D' = \{d'_k \mid 1 \le k \le n\}$. We can assume without loss of generality that $|D' \cap V(B^{i_1 i_2}_{n-2})| \ge |D' \cap V(B^{i_1\bar{i}_2}_{n-2})|$.

Step 2) Select two disjoint paths of length at most 2 between $\{s_1, s_2\}$ and two distinct nodes in $D' \cap V(B^{i_1 i_2}_{n-2})$ in $B^{i_1 i_2}_{n-2}$. We can assume without loss of generality that two paths $s_1 \rightsquigarrow d'_1$ and $s_2 \rightsquigarrow d'_2$ have been constructed.

Step 3) For each node d'_k $(k \in \{3, 4\})$ such that $d'_k \in V(B^{i_1\bar{i}_2}_{n-2})$, let d''_k be d'_k. If for every node d'_k $(k \in \{3, 4\})$ such that $d'_k \in V(B^{i_1 i_2}_{n-2})$ the node $d'^{(n-1)}_k$ is not in D', select for each such d'_k the edge $d'_k \to d'^{(n-1)}_k (= d''_k)$, and go to Step 4. Otherwise, we can assume without loss of generality that there exists $k \in \{3, 4\}$ with $d'_k \in V(B^{i_1 i_2}_{n-2})$ and $d'^{(n-1)}_k \in D'$. See Fig. 14.

In this configuration, first discard the paths $s_1 \rightsquigarrow d'_1$ and $s_2 \rightsquigarrow d'_2$ constructed in Step 2. Next, for either one of d_k $(k \in \{3, 4\})$, find a path $d_k \to d_k^{(x)} \to d_k^{(x,n)} (= d'_k)$ $(1 \le x \le n-1)$ such that the path does not include any destination nodes other than d_k, and select it. We can assume without loss of generality that $k = 3$. Then, select edges $d_k \to d_k^{(n)} (= d'_k)$ $(k \in \{1, 2, 4\})$. Let $D' = \{d'_1, d'_2, d'_3, d'_4\}$, and go back to Step 2.

Step 4) Select two disjoint paths of lengths at most 2 between $\{s_3, s_4\}$ and $\{d''_3, d''_4\}$ in $B^{i_1\bar{i}_2}_{n-2}$.

VI. CORRECTNESS AND COMPLEXITIES

In this section we prove the correctness and the complexities of Algorithms FTR and S2S. We assume that each node a $(\in V(B_n))$ is stored in a machine word and construction of an edge by constructing a neighbor node $a^{(i)}$ $(1 \le i \le n)$ requires $O(1)$ time.

A. ALGORITHM FTR

Let $T_0(f, n)$ and $L_0(f, n)$ represent the time complexity and the maximum path length of Algorithm FTR, respectively, if it is applied to construct a fault-free path in B_n with f (≥ 2) faulty nodes.

Lemma 4: In Case 1 Algorithm FTR constructs a fault-free path between s *and* d *of length at most* $L_0(\lfloor f/2 \rfloor, n-1) + 4$ *in* $T_0(\lfloor f/2 \rfloor, n-1) + O(n)$ *time.*

Proof: In Step 1 it takes $O(n)$ time to compare $|V(B^i_{n-1}) \cap F|$ and $|V(B^{\bar{i}}_{n-1}) \cap F|$. Also, it takes $T_0(\lfloor f/2 \rfloor, n-1)$ time to apply FTR recursively in B^i_{n-1}, and the maximum length of the path constructed by FTR is $L_0(\lfloor f/2 \rfloor, n-1)$. In Step 2 it takes $O(n)$ time to check if $s^{(n)} \notin F$. Selection of the path $s \to s^{(n)}$ or $s \to s^{(j)} \to s^{(j,n)}$ takes $O(n)$ time and the path length is at most 2. In Step 3 it takes $O(n)$ time to check if $d^{(n)} \notin F$. Selection of the path $d \to d^{(n)}$ or $d \to d^{(j)} \to d^{(j,n)}$ takes $O(n)$ time and the path length is at most 2. In Step 4 the recursive application of FTR takes $T_0(\lfloor f/2 \rfloor, n-1)$ time and the maximum length of the path constructed is $L_0(\lfloor f/2 \rfloor, n-1)$. Therefore, in total, FTR takes $T_0(\lfloor f/2 \rfloor, n-1) + O(n)$ time, and the maximum length of the constructed path is $L_0(\lfloor f/2 \rfloor, n-1) + 4$ in Case 1. □

Lemma 5: In Case 2 Algorithm FTR constructs a fault-free path between s *and* d *of length at most* $L_0(\lfloor f/2 \rfloor, n-1) + 2$ *in* $T_0(\lfloor f/2 \rfloor, n-1) + O(n)$ *time.*

Proof: It takes $O(n)$ time to compare $|V(B^i_{n-1}) \cap F|$ and $|V(B^{\bar{i}}_{n-1}) \cap F|$. In Step 1 it takes $O(n)$ time to check if $d^{(n)} \notin F$. Selection of the path $s \to s^{(n)}$ or $s \to s^{(j)} \to s^{(j,n)}$ takes $O(n)$ time and the path length is at most 2. In Step 2 the recursive application of FTR takes $T_0(\lfloor f/2 \rfloor, n-1)$ time and the maximum length of the constructed path is $L_0(\lfloor f/2 \rfloor, n-1)$. Therefore, in total, FTR takes $T_0(\lfloor f/2 \rfloor, n-1) + O(n)$ time, and the maximum length of the path is $L_0(\lfloor f/2 \rfloor, n-1) + 2$ in Case 2. □

Theorem 1: Algorithm FTR constructs a fault-free path between s *and* d *of length at most* $n + 3\lfloor \log_2 f \rfloor - 1$ *in* $O(n \log f)$ *time.*

Proof: It takes $O(1)$ time to check if $\boldsymbol{s}, \boldsymbol{d} \in B^i_{n-1}$ ($i \in \{0, 1\}$). After applying FTR $\lfloor \log_2 f \rfloor$ times, we can apply R in an $(n - \lfloor \log_2 f \rfloor)$-dimensional BC graph with at most one faulty node. In the final application of FTR, $BC^{\bar{i}}_{n-\lfloor \log_2 f \rfloor}$ contains at most one faulty node. Hence, at least one of the edges $\boldsymbol{s} \rightarrow \boldsymbol{s}^{(n)}$ and $\boldsymbol{d} \rightarrow \boldsymbol{d}^{(n)}$ is fault-free in Case 1. Therefore, from Lemmas 3, 4 and 5, we can have $T_0(f, n) = O(n \log f)$ and $L_0(f, n) = 4(\lfloor \log_2 f \rfloor - 1) + 3 + (n - \lfloor \log_2 f \rfloor) = n + 3\lfloor \log_2 f \rfloor - 1$. □

B. ALGORITHM S2S

Let $T_1(p, n)$ and $L_1(p, n)$ represent the time complexity and the maximum path length of Algorithm S2S, respectively, if it is applied to construct p disjoint paths in B_n. The following lemmas are proved with mathematical induction.

Lemma 6: In Case 1 Algorithm S2S constructs p disjoint paths between S and D. Their lengths are at most $L_1(p, n-1)$ *if* $p < n$*, and* $\max\{L_1(p-1, p-1), p+1\}$ *if* $p = n$*. The time complexity of the algorithm is* $T_1(p, n-1) + O(1)$ *if* $p < n$*, and* $T_1(p-1, p-1) + L_1(p-1, p-1) \times O(p)$ *if* $p = n$*.*

Proof: In Step 1 it takes $O(1)$ to check if $p < n$. If $p < n$, S2S constructs p disjoint paths of length at most $L_1(p, n-1)$ in $T_1(p, n-1) + O(1)$ time from the hypothesis of the induction. In Step 2 applying S2S in B^i_{p-1} recursively takes $T_1(p-1, p-1)$ time to construct $(p-1)$ disjoint paths whose lengths are at most $L_1(p-1, p-1)$ from the hypothesis of the induction. In Step 3 it takes $L_1(p-1, p-1) \times O(p)$ time to check if $\boldsymbol{s}_p$ is included in one of the paths constructed in Step 2. It takes $O(1)$ time to discard the sub path $\boldsymbol{s}_x \rightsquigarrow \boldsymbol{s}_p$, and exchange the indices of $\boldsymbol{s}_x$ and $\boldsymbol{s}_p$. In Step 4 it takes $L_1(p-1, p-1) \times O(p)$ time to check if $\boldsymbol{d}_p$ is included in one of the paths constructed in Steps 2 and 3. It takes $O(1)$ time to discard the sub path $\boldsymbol{d}_p \rightsquigarrow \boldsymbol{d}_y$, and exchange the indices of $\boldsymbol{d}_y$ and $\boldsymbol{d}_p$. In Step 5 it takes $O(1)$ time to check if $\boldsymbol{s}_p = \boldsymbol{d}_p$. Selection of the edges $\boldsymbol{s}_p \rightarrow \boldsymbol{s}^{(n)}_p$ and $\boldsymbol{d}_p \rightarrow \boldsymbol{d}^{(n)}_p$ takes $O(1)$ time and the path lengths are both 1. In Step 6 it takes $O(p)$ time to construct a path between $\boldsymbol{s}'_p$ and $\boldsymbol{d}'_p$ of length at most $p-1$ from Lemma 3. The constructed path $\boldsymbol{s}_p \rightsquigarrow \boldsymbol{d}_p$ is outside of B^i_{n-1} except for $\boldsymbol{s}_p$ and $\boldsymbol{d}_p$. Hence, it is disjoint from other constructed paths. Therefore, in total, S2S construct p disjoint paths in Case 1 by taking $T_1(p, n-1) + O(1)$ time, and the maximum path length is $L_1(p, n-1)$ if $p < n$, and by taking $T_1(p-1, p-1) + L_1(p-1, p-1) \times O(p)$ time, and the maximum path length is $\max\{L_1(p-1, p-1), p+1\}$ if $p = n$. □

Lemma 7: In Case 2 Algorithm S2S constructs p disjoint paths between S and D. Their lengths are at most $\max\{L_1(l, n-1), L_1(p-l, n-1) + 2\}$*. The time complexity of the algorithm is* $T_1(l, n-1) + T_1(p-l, n-1) + \{L_1(l, n-1)\}^2 \times O(np^3)$*.*

Proof: In Step 1 applying S2S recursively in B^i_{n-1} takes $T_1(l, n-1)$ time to construct l disjoint paths whose lengths are at most $L_1(l, n-1)$ from the induction hypothesis. In Step 2 it takes $L_1(l, n-1) \times O(l)$ time to check if each $\boldsymbol{d}_k$ ($l+1 \leq k \leq m$) is included in one of the paths constructed in Step 1. It takes $O(1)$ time to discard the sub path $\boldsymbol{d}_k \rightsquigarrow \boldsymbol{d}_y$, and exchange the indices of $\boldsymbol{d}_y$ and $\boldsymbol{d}_k$. Hence, in total, it takes $L_1(l, n-1) \times O(l) \times O(m-l) = L_1(l, n-1) \times O(p^2)$ time in Step 2. In Step 3 it takes $O(p-m)$ time to check if $\boldsymbol{d}^{(n)}_k \notin D$ for each $\boldsymbol{d}_k$ ($l+1 \leq k \leq m$). If $\boldsymbol{d}^{(n)}_k \notin D$, it takes $O(1)$ time to construct the path $\boldsymbol{d}_k \rightarrow \boldsymbol{d}^{(n)}_k$ of length 1. Otherwise, it takes $(L_1(l, n-1) \times O(l) + O(m-l) + O(p-m)) \times O(n)$ time to check if there is j ($1 \leq j \leq p-1$) such that $\boldsymbol{d}^{(j)}_k$ is not included in D or the paths constructed in Steps 1, 2, and 3 so far, and $\boldsymbol{d}^{(j,n)}_k \notin D$. If such j exists, it takes $O(1)$ time to construct the path $\boldsymbol{d}_k \rightarrow \boldsymbol{d}^{(j)}_k \rightarrow \boldsymbol{d}^{(j,n)}_k$ of length 2. If such j does not exist, it takes $L_1(l, n-1) \times O(l) \times O(n)$ time to find the path $\boldsymbol{s}_x \rightsquigarrow \boldsymbol{d}_y$ and $\boldsymbol{d}^{(j_1)}_k$. It takes $O(1)$ time to discard the sub path $\boldsymbol{d}^{(j_1)}_k \rightsquigarrow \boldsymbol{d}^{(j_2)}_k \rightsquigarrow \boldsymbol{d}_y$, add the edge $\boldsymbol{d}^{(j_1)}_k \rightarrow \boldsymbol{d}_k$, and exchange the indices of $\boldsymbol{d}_y$ and $\boldsymbol{d}_k$. From Lemma 1, there are at least two hops between $\boldsymbol{d}^{(j_1)}_k$ and $\boldsymbol{d}^{(j_2)}_k$. Hence, the length of the path $\boldsymbol{s}_x \rightsquigarrow \boldsymbol{d}_y$ is shortened by at least one hop. So, this process will terminate after at most $L_1(l, n-1) \times O(l)$ repetitions. Hence, in total, it takes $(L_1(l, n-1) \times O(l) + O(m-l) + O(p-m)) \times O(n) \times L_1(l, n-1) \times O(l) \times O(m-l) = \{L_1(l, n-1)\}^2 \times O(np^3)$ time to construct the $(m-l)$ disjoint paths of lengths at most 2 in Step 3. In Step 4 applying S2S recursively in $B^{\bar{i}}_{n-1}$ to construct $(p-l)$ disjoint paths takes $T_1(p-l, n-1)$ time, and the maximum path length is $L_1(p-l, n-1)$ from the hypothesis of the induction. Therefore, in total, S2S constructs p disjoint paths in Case 2 by taking $T_1(l, n-1) + T_1(p-l, n-1) + \{L_1(l, n-1)\}^2 \times O(np^3)$ time, and the maximum path length is $\max\{L_1(l, n-1), L_1(p-l, n-1) + 2\}$. □

Lemma 8: In Case 3-1 Algorithm S2S constructs p disjoint paths between S and D. Their lengths are at most $L_1(p, n-1) + 1$ *if* $p < n$*, and* $\max\{L_1(p-1, p-1) + 1, p+1\}$ *if* $p = n$*. The time complexity of the algorithm is* $T_1(p, n-1) + O(p)$ *if* $p < n$*, and* $T_1(p-1, p-1) + L_1(p-1, p-1) \times O(p)$ *if* $p = n$*.*

Proof: If $p < n$, selection of edges $\boldsymbol{d}_k \rightarrow \boldsymbol{d}^{(n)}_k$ ($1 \leq k \leq p$) takes $O(p)$ time. S2S recursively applied in $B^{\bar{i}_1}_{n-1}$ constructs p disjoint paths from the hypothesis of the induction. The path lengths are at most $L_1(p, n-1) + 1$, and the time complexity is $T_1(p, n-1) + O(p)$. In the rest of this proof, we assume that $p = n$. In Step 1 it takes $O(p)$ time to construct the paths $\boldsymbol{s}_k \rightarrow \boldsymbol{s}^{(n)}_k$ ($1 \leq k \leq p-1$) of length 1. Because the paths are constructed by a bijection, they are disjoint. In Step 2 applying S2S recursively in $B^{\bar{i}_1}_{n-1}$ to construct $(p-1)$ disjoint paths of length at most $L_1(p-1, p-1)$ takes $T_1(p-1, p-1)$ time from the induction hypothesis. In Step 3 it takes $L_1(p-1, p-1) \times O(p)$ time to check if $\boldsymbol{d}_p$ is included in one of the paths constructed in Step 2. It takes $O(1)$ time to discard the sub path $\boldsymbol{d}_p \rightsquigarrow \boldsymbol{d}_y$ and exchange the indices of $\boldsymbol{d}_y$ and $\boldsymbol{d}_p$ if $\boldsymbol{d}_p$ is included in the path $\boldsymbol{s}'_x \rightsquigarrow \boldsymbol{d}_y$. In Step 4 it takes $O(1)$ time to construct the path $\boldsymbol{d}_p \rightarrow \boldsymbol{d}^{(n)}_p$ of length 1. If $\boldsymbol{d}^{(n)}_p \in (S \setminus \{\boldsymbol{s}_p\} \cap V(B^{i_1}_{n-1}))$, $\boldsymbol{d}^{(n)}_p$ must be included in the paths constructed in Step 2, and it is a contradiction. Hence, $\boldsymbol{d}^{(n)}_p \notin (S \setminus \{\boldsymbol{s}_p\} \cap V(B^{i_1}_{n-1}))$, and the path $\boldsymbol{d}_p \rightarrow \boldsymbol{d}^{(n)}_p$ is disjoint from other paths constructed so far. It takes $O(1)$ time to check if $\boldsymbol{d}^{(n)}_p = \boldsymbol{s}_p$. In Step 5 it takes $O(1)$ time to check if $s_p \in V(B^{i_1 i_2}_{n-2})$ and construct the path $s_p \to s_p^{(n-1)}$ of length 1. Because no source or destination node exists in $B^{i_1 i_2}_{n-2}$, the path $s_p \to s_p^{(n-1)}$ is disjoint from other paths constructed in Step 1. In Step 6 it takes $O(1)$ time to check if $d_p^{(n)} \in V(B^{i_1 i_2}_{n-2})$ and construct the path $d_p^{(n)} \to d_p^{(n,n-1)}$ of length 1. Because neither a source node, nor a destination node, nor a path from either exists in $B^{i_1 \bar{i}_2}_{n-2}$ except for the node s'_p, the path $d_p^{(n)} \to d_p^{(n,n-1)}$ is disjoint from other paths constructed in Steps 1 to 4 except for the one from s_p. In Step 7 from Lemma 3, applying R in $B^{i_1 \bar{i}_2}_{n-2}$ takes $O(p)$ time and the length of the constructed path is at most $p-2$. The path from s_p to d_p is included in $B^{i_1 \bar{i}_2}_{n-2}$ except for the nodes s_p, $d_p^{(n)}$, and d_p, which are disjoint from other paths. Hence, the path is disjoint from other paths. Therefore, in total, S2S constructs p disjoint paths in Case 3-1 by taking $T_1(p, n-1)+O(p)$ time, and the maximum path length is $L_1(p, n-1)+1$ if $p < n$, and by taking $T_1(p-1, p-1)+L_1(p-1, p-1) \times O(p)$ time, and the maximum path length is $\max\{L_1(p-1, p-1)+1, p+1\}$ if $p = n$. □

Lemma 9: In Case 3-2 Algorithm S2S constructs p disjoint paths between S and D. Their lengths are at most $L_1(p, n-1)+1$ *if* $p < n$, *and* $\max\{L_1(p-1, p-1)+1, p+3\lfloor\log_2(p+1)\rfloor - 4\}$ *if* $p = n$. *The time complexity of the algorithm is* $T_1(p, n-1) + O(p)$ *if* $p < n$, *and* $T_1(p-1, p-1) + L_1(p-1, p-1) \times O(p) + O(p \log p)$ *if* $p = n$.

Proof: As in the beginning of the previous proof, if $p < n$, selection of edges $d_k \to d_k^{(n)}$ $(1 \le k \le p)$ takes $O(p)$ time. S2S recursively applied in $B^{\bar{i}_1}_{n-1}$ constructs p disjoint paths from the hypothesis of the induction. The path lengths are at most $L_1(p, n-1) + 1$, and the time complexity is $T_1(p, n-1) + O(p)$. In the rest of this proof, we assume that $p = n$. In Step 1 it takes $O(p)$ time to check if there is a source node $s_x (\in V(B^{i_1 i_2}_{n-2}))$ such that $s_x^{(n-1)} \notin S$, and it takes $O(1)$ time to exchange the indices of s_x and s_p. In Step 2 it takes $O(p)$ time to select the paths $s_k \to s_k^{(n)}$ $(1 \le k \le p-1)$ of length 1. Because the paths are constructed by a bijection, they are disjoint. In Step 3 applying S2S recursively in $B^{\bar{i}_1}_{n-1}$ to construct $(p-1)$ disjoint paths of length at most $L_1(p-1, p-1)$ takes $T_1(p-1, p-1)$ time from the induction hypothesis. In Step 4 it takes $L_1(p-1, p-1) \times O(p)$ time to check if d_p is included in one of the paths constructed in Step 3. It takes $O(1)$ time to discard the sub path $d_p \rightsquigarrow d_y$ if d_p is included in the path $s'_x \rightsquigarrow d_y$. In Step 5 it takes $O(1)$ time to select the path $d_p \to d_p^{(n)}$ of length 1. If $d_p^{(n)} \in (S \setminus \{s_p\} \cap V(B^{i_1}_{n-1}))$, $d_p^{(n)}$ must be included in the paths constructed in Step 3, and it is a contradiction. Hence, $d_p^{(n)} \notin (S \setminus \{s_p\} \cap V(B^{i_1}_{n-1}))$, and the path $d_p \to d_p^{(n)}$ is disjoint from other paths constructed so far. It takes $O(1)$ time to check if $d_p^{(n)} = s_p$. In Step 6 it takes $O(p)$ time to check if $d_p^{(n)} \in V(B^{i_1 \bar{i}_2}_{n-2})$ and $s_p^{(n-1)} \notin S$, and select the path $s_p \to s_p^{(n-1)}$ of length 1. From the assumption that $|S \cap V(B^{i_1 \bar{i}_2}_{n-2})| \le |S \cap V(B^{(i_1 i_2)}_{n-2})|$, there are at most $\lfloor p/2 \rfloor$ source nodes in $B^{i_1 \bar{i}_2}_{n-2}$. If $n \ge 4$, $\lfloor p/2 \rfloor < n-2$ holds for any $p \le n$. If $n = p = 3$, $s_p^{(n-1)} = d_p^{(n)}$. Hence, FTR can construct a disjoint path from $s_p^{(n-1)}$ to $d_p^{(n)}$. From Theorem 1, applying FTR in $B^{i_1 \bar{i}_2}_{n-2}$ to construct a fault-free path $s_p^{(n-1)} \rightsquigarrow d_p^{(n)}$ of length at most $p+3\lfloor\log_2 p\rfloor - 6(\ge p-2+3\lfloor\log_2 \lfloor p/2 \rfloor\rfloor - 1)$ takes $O(p \log p)$ time. In Step 7 it takes $O(1)$ time to check if $d_p^{(n)} \in V(B^{i_1 i_2}_{n-2})$ and construct the path $d_p^{(n)} \to d_p^{(n,n-1)}$ of length 1. If $d_p^{(n)} \in V(B^{i_1 \bar{i}_2}_{n-2})$, $s_p^{(n-1)} \in S$ from Step 6. Thus, $|S \cap V(B^{i_1 \bar{i}_2}_{n-2})| = |S \cap V(B^{i_1 i_2}_{n-2})|$, and for each source node s_x in $V(B^{i_1 i_2}_{n-2})$, $s_x^{(n-1)} \in S$. Hence, $d_p^{(n,n-1)} \notin S$. Therefore, the path $d_p^{(n)} \to d_p^{(n,n-1)}$ is disjoint from other paths. In Step 8 because $|S \cap V(B^{i_1 i_2}_{n-2})| \le p-2$, there are at most $(p-3)$ source nodes except for s_p in $B^{i_1 i_2}_{n-2}$. Hence, FTR can construct a disjoint path from s_p to d'_p. From Theorem 1, applying FTR in $B^{i_1 i_2}_{n-2}$ to construct a fault-free path $s_p \rightsquigarrow d'_p$ of length at most $p + 3\lfloor\log_2(p+1)\rfloor - 6(\ge p-2+3\lfloor\log_2 \lceil p/2 \rceil\rfloor - 1)$ takes $O(p \log p)$ time. Therefore, in total, S2S constructs p disjoint paths in Case 3-2 by taking $T_1(p, n-1) + O(p)$ time, and the maximum path length is $L_1(p, n-1) + 1$ if $p < n$, and by taking $T_1(p-1, p-1) + L_1(p-1, p-1) \times O(p) + O(p \log p)$ time, and the maximum path length is $\max\{L_1(p-1, p-1) + 1, p + 3\lfloor\log_2(p+1)\rfloor - 4\}$ if $p = n$. □

Lemma 10: In Case 3-3 Algorithm S2S constructs four disjoint paths between S and D. Their lengths are at most 5. The time complexity of the algorithm is $O(1)$.

Proof: In Step 1 it takes $O(1)$ time to select four paths $d_k \to d_k^{(n)}$ $(1 \le k \le 4)$ of length 1. Let $D' = \{d_k^{(n)} \mid 1 \le k \le 4\}$. Then, we can assume without loss of generality that $|D' \cap V(B^{i_1 \bar{i}_2}_{n-2})| \le |D' \cap V(B^{i_1 i_2}_{n-2})|$ and $d_1^{(n)}, d_2^{(n)} \in V(B^{i_1 \bar{i}_2}_{n-2})$. In Step 2 it takes $O(1)$ time to construct two disjoint paths of lengths at most 2 between $\{s_1, s_2\}$ and two nodes in $D' \cap V(B^{i_1 \bar{i}_2}_{n-2})$. We can assume without loss of generality that $s_1 \rightsquigarrow d_1^{(n)}$ and $s_2 \rightsquigarrow d_2^{(n)}$ are constructed. In Step 3 it takes $O(1)$ time to select at most two paths $d_k^{(n)} \to d_k^{(n,n-1)}$ $(k \in \{3, 4\})$ of length 1. If the path $d_k^{(n)} \to d_k^{(n,n-1)}$ cannot be selected because $d_k^{(n,n-1)} \in D'$, it takes $O(1)$ time to construct a detour path $d_k \to d_k^{(x)} \to d_k^{(x,n)}$ $(k \in \{3, 4\})$ of length 2. After the detour path is constructed, the process restarts from Step 2. However, in this configuration, another detour path is never needed in Step 3. Hence, Steps 2 and 3 are executed at most twice. In Step 4 it takes $O(1)$ time to construct two disjoint paths of lengths at most 2 between $\{s_3, s_4\}$ and $\{d''_3, d''_4\}$. Therefore, in Case 3-3, S2S takes $O(1)$ time to construct four disjoint paths of length at most 5. □

Theorem 2: Algorithm S2S constructs p disjoint paths between S and D in B_n. *Their lengths are at most* $L_1(p, n) = n + p - 1$, *and the time complexity is* $T_1(p, n) = O(n^3 p^4)$.

Proof: From Lemmas 6 to 10, it is proved that S2S constructs p disjoint paths between S and D.

First, let us consider the case that $p = n$. Then, from Lemmas 6 to 9, $L_1(p, p) = \max\{L_1(p-1, p-1), p+1\}$ in Case 1, $L_1(p, p) = \max\{L_1(l, p-1), L_1(p-l, p-1)+2\}$ in Case 2, $L_1(p, p) = \max\{L_1(p-1, p-1)+1, p+1\}$ in Case 3-1, and $L_1(p, p) = \max\{L_1(p-1, p-1)+1, p+3\lfloor\log_2(p+1)\rfloor - 4\}$ in Case 3-2. Thus, by assigning 1 to l in Case 2, $L_1(p, p) = L_1(p-1, p-1) + 2 = L_1(1, 1) + 2(p-1) = 2p-1$ holds. Also, from Lemmas 6 to 9, $T_1(p, p) = T_1(p-1, p-1) +$

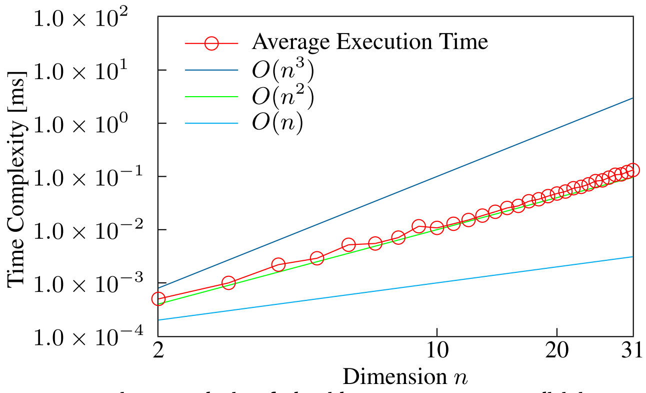

FIGURE 15. Time complexity of Algorithm S2S to construct n disjoint paths in n-dimensional locally twisted cubes.

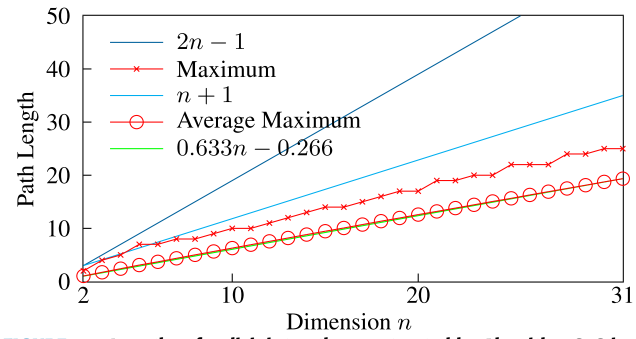

FIGURE 16. Lengths of n disjoint paths constructed by Algorithm S2S in n-dimensional locally twisted cubes. Note that the green line is almost identical to the average maximum and therefore hidden in the graph.

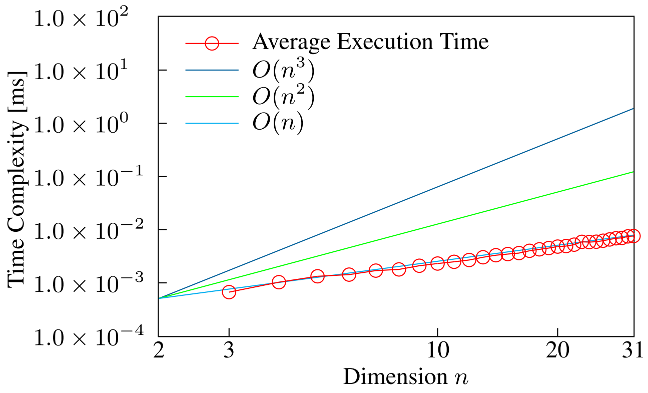

FIGURE 17. Time complexity of Algorithm FTR to construct a fault-free path in an n-dimensional locally twisted cube with $(n-1)$ faulty nodes.

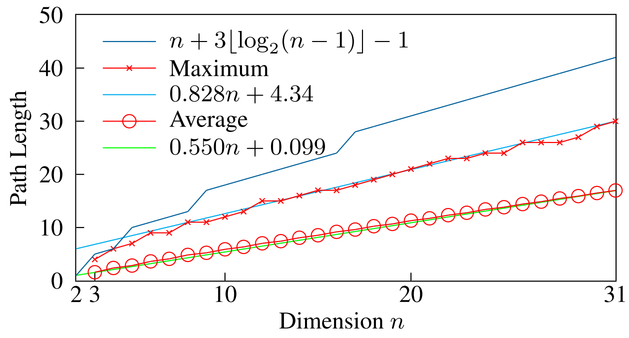

FIGURE 18. Length of the fault-free path constructed by Algorithm FTR in an n-dimensional locally twisted cube with $(n-1)$ faulty nodes. Note that the green line is almost identical to the average and therefore hidden in the graph.

$L_1(p-1,p-1)\times O(p)$ in Case 1, $T_1(p,p)=T_1(l,p-1)+T_1(p-l,p-1)+\{L_1(l,p-1)\}^2\times O(p^4)$ in Case 2, $T_1(p,p)=T_1(p-1,p-1)+L_1(p-1,p-1)\times O(p)$ in Case 3-1, and $T_1(p,p)=T_1(p-1,p-1)+L_1(p-1,p-1)\times O(p)+O(p\log p)$ in Case 3-2. Because $L_1(p-1,p-1)=O(p)$, $T_1(p,p)=T_1(p-1,p-1)+O(p^6)=O(p^7)$ holds.

Next, let us consider the case that $p<n$. Then, from Lemmas 6 to 9, $L_1(p,n)=L_1(p,n-1)$ in Case 1, $L_1(p,n)=\max\{L_1(l,n-1),L_1(p-l,n-1)+2\}$ in Case 2, $L_1(p,n)=L_1(p,n-1)+1$ in Case 3-1, and $L_1(p,n)=L_1(p,n-1)+1$ in Case 3-2. Thus, by assigning 1 to l in Case 2, $L_1(p,n)=\max\{L_1(p-1,n-1)+2,L_1(p,n-1)+1\}=\max\{L_1(1,n-p+1)+2(p-1),L_1(p,p)+(n-p)\}=\max\{(n-p+1)+2(p-1),(2p-1)+(n-p)\}=n+p-1$ holds. Also, from Lemmas 6 to 9, $T_1(p,n)=T_1(p,n-1)+O(1)$ in Case 1, $T_1(p,n)=T_1(l,n-1)+T_1(p-l,n-1)+\{L_1(l,n-1)\}^2\times O(np^3)$ in Case 2, $T_1(p,n)=T_1(p,n-1)+O(p)$ in Case 3-1, and $T_1(p,n)=T_1(p,n-1)+O(p)$ in Case 3-2. Because $L_1(l,n-1)=O(n)$, $T_1(p,n)=\max\{T_1(l,n-1)+T_1(p-l,n-1)+O(n^3p^3),T_1(p,n-1)+O(p)\}=\max\{O(np)+O(n^3p^4),O(np+p^7)\}=O(n^3p^4)$ holds. Finally, considering Lemma 10 for the exceptional case, that is, Case 3-3, this theorem is proved. □

VII. COMPUTER EXPERIMENT

To observe the average behavior of Algorithm S2S, we have selected the locally twisted cube as an example of a BC graph and implemented S2S by using the programming language Racket. The locally twisted cube is selected because any disjoint paths problem has not been solved for it. Hence, our algorithm provides the solutions to all of them. We have conducted an experiment on a computer with the AMD Ryzen 9 5900HX with Radeon Graphics (3.30 GHz) processor, 16.0 GB memory, and the Microsoft Windows 11 Home Edition operating system.

The experiment is carried out in three steps as follows:

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

Step 2) In an n-dimensional locally twisted cube, select n distinct source nodes and n distinct destination nodes randomly.

Step 3) Apply S2S, and measure the maximum path length, the sum of path lengths, and the sum of execution time.

Figure 15 shows the average execution time, and Figure 16 shows the maximum path lengths and the average maximum path lengths.

The obtained results regarding the time complexity (see Fig. 15) clearly show that, as the dimension n of the network increases, the average time complexity of the proposed algorithm is $O(n^2)$. This is to be compared with the theoretical worst-case time complexity that has been previously established (see Theorem 2): the gap between the worst-case time complexity $O(n^3p^4)$ and the experimentally obtained average time complexity is a good indicator of the performance of the algorithm.

A similar discussion can be made with respect to the path length analysis: the experimental results (see Fig. 16) show that on average, as the dimension n of the network increases, the maximum path length stays well below $n + 1$, approximately $0.633n - 0.266$. This is to be compared with the theoretical maximum path length previously calculated (see Theorem 2) of $2n - 1$ (in this experiment, $p = n$). So, the difference between the experimentally obtained maximum path length and the theoretical one is yet another strong indicator of the performance of our algorithms.

VIII. CONCLUSION

In this paper we proposed a polynomial-order time algorithm for the set-to-set disjoint paths problem in BC graphs. Its time complexity is $O(n^3p^4)$, and the maximum path length is $n + p - 1$ if the algorithm is applied to construct p $(\leq n)$ disjoint paths in an n-dimensional BC graph. We also conducted a computer experiment with an n-dimensional locally twisted cube to construct n disjoint paths and showed that the average execution time is $O(n^2)$ and the maximum path lengths on average are approximately $0.633n - 0.266$.

Future works include the theoretical analysis of the maximum path length of the algorithm as well as its average performance. Also, improvement of the algorithm to construct shorter paths in smaller execution time is also of interest. Moreover, we wish to extend our algorithm so that it can be applied to the k-ary n-cube [53], [54], the hierarchical topologies such as the hierarchical cubic network [55], [56] and the cube-connected cycles [52], [57], and the product-based topologies such as the DQcube [58], [59].

APPENDIX
COMPUTER EXPERIMENT ON ALGORITHM FTR

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

Step 1) For each n between 3 to 31, execute Steps 2 to 4 for 100,000 times.

Step 2) In an n-dimensional locally twisted cube, select $(n-1)$ distinct faulty nodes randomly.

Step 3) Select a source node and a destination node from non faulty nodes randomly.

Step 4) Apply FTR, and measure the maximum path length, the sum of path lengths, and the sum of execution time.

Figure 17 shows the average execution time, and Figure 18 shows the maximum and average path lengths.

Figure 17 shows that the average execution time of Algorithm FTR in an n-dimensional locally twisted cube with $(n-1)$ faulty nodes is $O(n)$ time. From Figure 18, the lengths of paths constructed by Algorithm FTR have not attained the theoretical upper bound $n + 3\lfloor \log_2(n-1) \rfloor - 1$ except for the case $n = 4$. Also, the maximum and average lengths of paths constructed by Algorithm FTR are both almost linear to n.

REFERENCES

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

[2] J. Fan and L. He, "BC interconnection networks and their properties (in Chinese)," *Chin. J. Comput.*, vol. 26, no. 1, pp. 84–90, Jan. 2003.

[3] P. A. J. Hilbers, M. R. Koopman, and J. L. A. van de Snepscheut, "The twisted cube," in *Parallel Architectures on 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

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

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

[6] X. Yang, D. J. Evans, and G. M. Megson, "The locally twisted cubes," *Int. J. Comput. Math.*, vol. 82, no. 4, pp. 401–413, Apr. 2005.

[7] 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.

[8] X. Wang, J. Liang, D. Qi, and W. Lin, "The twisted crossed cube," *Concurrency Comput., Pract. Exper.*, vol. 28, no. 5, pp. 1507–1526, Apr. 2016.

[9] J. Fan and X. Jia, "Edge-pancyclicity and path-embeddability of bijective connection graphs," *Inf. Sci.*, vol. 178, no. 2, pp. 340–351, Jan. 2008.

[10] J. Fan, X. Jia, B. Cheng, and J. Yu, "An efficient fault-tolerant routing algorithm in bijective connection networks with restricted faulty edges," *Theor. Comput. Sci.*, vol. 412, no. 29, pp. 3440–3450, Jul. 2011. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0304397511001344

[11] M. Zhang, J. Meng, W. Yang, and Y. Tian, "Reliability analysis of bijective connection networks in terms of the extra edge-connectivity," *Inf. Sci.*, vol. 279, pp. 374–382, Sep. 2014. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0020025514004228

[12] L. Guo and C.-W. Lee, "Reliability analysis of the bijective connection networks for components," *Mathematics*, vol. 7, no. 6, p. 546, Jun. 2019. [Online]. Available: https://www.mdpi.com/2227-7390/7/6/546

[13] K. Kaneko, "Node-disjoint paths problems in directed bijective connection graphs," *IEICE Trans. Inf. Syst.*, vol. 103, no. 1, pp. 93–100, Jan. 2020.

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

[15] 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.

[16] A. Bossard and K. Kaneko, "Set-to-set disjoint paths routing in hierarchical cubic networks," *Comput. J.*, vol. 57, no. 2, pp. 332–337, Feb. 2014.

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

[18] Q.-P. Gu, S. Okawa, and S. Peng, "Set-to-set fault tolerant routing in hypercudes," *IEICE Trans. Fundamentals Electron., Commun. Comput. Sci.*, vol. 79, no. 4, pp. 483–488, Apr. 1996.

[19] Q.-P. Gu and S. Peng, "Set-to-set fault tolerant routing in star graphs," *IEICE Trans. Inf. Syst.*, vol. 79, no. 4, pp. 282–289, Apr. 1996.

[20] 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.

[21] K. Kaneko and A. Bossard, "A set-to-set disjoint paths routing algorithm in tori," *Int. J. Netw. Comput.*, vol. 7, no. 2, pp. 173–186, 2017.

[22] 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

[23] 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.

[24] 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.

[25] 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.

[26] Y. Hamada, F. Bao, A. Mei, and Y. Igarashi, "Nonadaptive fault-tolerant file transmission in rotator graphs," *IEICE Trans. Fundamentals*, vol. 79, no. 4, pp. 477–482, Apr. 1996.
[27] K. Kaneko and N. Sawada, "An algorithm for node-to-node disjoint paths problem in burnt pancake graphs," *IEICE Trans. Inf. Syst.*, vol. 90, no. 1, pp. 306–313, Jan. 2007.
[28] D. Kocík and K. Kaneko, "Node-to-node disjoint paths problem in a Möbius cube," *IEICE Trans. Inf. Syst.*, vol. 100, no. 8, pp. 1837–1843, Aug. 2017.
[29] 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.
[30] 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.
[31] 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.
[32] 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.
[33] Y. Suzuki and K. Kaneko, "An algorithm for node-disjoint paths in pancake graphs," *IEICE Trans. Inf. Syst.*, vol. 86, no. 3, pp. 610–615, Mar. 2003.
[34] Y. Suzuki and K. Kaneko, "The container problem in bubble-sort graphs," *IEICE Trans. Inf. Syst.*, vol. 91, no. 4, pp. 1003–1009, Apr. 2008.
[35] R.-Y. Wu, G.-H. Chen, Y.-L. Kuo, and G. J. Chang, "Node-disjoint paths in hierarchical hypercube networks," *Inform. Sci.*, vol. 177, pp. 4200–4207, Oct. 2007.
[36] A. Bossard and K. Kaneko, "A node-to-set disjoint paths routing algorithm in torus-connected cycles," *ISCA Int. J. Comput. their Appl.*, vol. 22, no. 1, pp. 22–30, Jan. 2015.
[37] 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.
[38] K. Kaneko and Y. Suzuki, "An algorithm for node-to-set disjoint paths problem in rotator graphs," *IEICE Trans. Inf. Syst.*, vol. 84, no. 9, pp. 1155–1163, Sep. 2001.
[39] K. Kaneko, "An algorithm for node-to-set disjoint paths problem in burnt pancake graphs," *IEICE Trans. Inf. Syst.*, vol. 86, no. 12, pp. 2588–2594, Dec. 2003.
[40] D. Kocík, Y. Hirai, and K. Kaneko, "Node-to-set disjoint paths problem in a Möbius cube," *IEICE Trans. Inf. Syst.*, vol. 99, no. 3, pp. 708–713, Mar. 2016.
[41] C.-N. Lai, "An efficient construction of one-to-many node-disjoint paths in folded hypercubes," *J. Parallel Distrib. Comput.*, vol. 74, no. 4, pp. 2310–2316, Apr. 2014.
[42] 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.
[43] 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.
[44] Y. Li, S. Peng, and W. Chu, "Node-to-set disjoint-paths routing in recursive dual-net," *Int. J. Netw. Comput.*, vol. 1, no. 2, pp. 178–190, Jul. 2011.
[45] 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.
[46] Y. Saad and M. H. Schultz, "Topological properties of hypercubes," *IEEE Trans. Comput.*, vol. COM-37, no. 7, pp. 867–872, Jul. 1988.
[47] A. Bossard and K. Kaneko, "Time optimal node-to-set disjoint paths routing in hypercubes," *J. Inf. Sci. Eng.*, vol. 30, no. 4, pp. 1087–1093, Jul. 2014.
[48] 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.
[49] P. D. Kulasinghe, "Connectivity of the crossed cube," *Inf. Process. Lett.*, vol. 61, no. 4, pp. 221–226, Feb. 1997.
[50] K. Kaneko, "An algorithm for set-to-set disjoint paths problem in a Möbius cube," in *Proc. 2017 Int. Conf. Parallel Distrib. Process. Techn. Appl.*, Jul. 2017, pp. 39–45.
[51] H. Nagashima, K. Mouri, and K. Kaneko, "Node-to-node disjoint paths in twisted crossed cubes," in *Proc. 10th Int. Conf. Adv. Inf. Technol. (IAIT)*, Dec. 2018, pp. 5:1–5:8.
[52] F. P. Preparata and J. Vuillemin, "The cube-connected cycles: A versatile network for parallel computation," *Commun. ACM*, vol. 24, no. 5, pp. 300–309, May 1981.
[53] J. Quadras and S. S. Solomon, "Embedding of the folded hypercubes into tori," *Math. Comput. Sci.*, vol. 9, no. 2, pp. 177–183, Jun. 2015, doi: 10.1007/s11786-015-0223-3.
[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. Ghose and K. R. Desai, "Hierarchical cubic networks," *IEEE Trans. Parallel Distrib. Syst.*, vol. 6, no. 4, pp. 427–435, Apr. 1995.
[56] S. Zhou, S. Song, X. Yang, and L. Chen, "On conditional fault tolerance and diagnosability of hierarchical cubic networks," *Theor. Comput. Sci.*, vol. 609, pp. 421–433, Jan. 2016. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0304397515009366
[57] H. Habibian and A. Patooghy, "Fault-tolerant routing methodology for hypercube and cube-connected cycles interconnection networks," *J. Supercomput.*, vol. 73, no. 10, pp. 4560–4579, Oct. 2017.
[58] R.-W. Hung, "DVcube: A novel compound architecture of disc-ring graph and hypercube-like graph," *Theor. Comput. Sci.*, vol. 498, pp. 28–45, Aug. 2013. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0304397513004556
[59] M. Lv, S. Zhou, J. Liu, X. Sun, and G. Lian, "Fault diagnosability of DQcube under the PMC model," *Discrete Appl. Math.*, vol. 259, pp. 180–192, Apr. 2019. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0166218X19300083

KEIICHI KANEKO (Member, IEEE) received the B.E., M.E., and Ph.D. (Eng.) degrees 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, fault-tolerant systems, and pedagogical systems. He is a member of ACM, IEEE CS, IEICE, IPSJ, and JSSST.

ANTOINE BOSSARD received the B.S. and M.S. degrees from the Université de Caen Basse-Normandie, France, in 2005 and 2007, respectively, and the Ph.D. degree from the Tokyo University of Agriculture and Technology, Japan, in 2011.

He is currently an Associate Professor with the Graduate School of Science, Kanagawa University, Japan. For several years, he has been conducting research regarding Chinese characters and their processing by computer systems. His research interests include graph theory, interconnection networks, and dependable systems. He is a member of ACM, ISCA, and TUG.

FREDERICK C. HARRIS, JR. received the B.S. and M.S. degrees in mathematics and educational administration from Bob Jones University, Greenville, SC, USA, in 1986 and 1988, respectively, and the M.S. and Ph.D. degrees in computer science from Clemson University, Clemson, SC, USA, in 1991 and 1994, respectively.

He is currently a Professor with the Department of Computer Science and Engineering and the Director of the High Performance Computation and Visualization Laboratory, University of Nevada, Reno, USA. He is also the Nevada State EPSCoR Director and the Project Director for Nevada NSF EPSCoR. He has published more than 290 peer-reviewed journals and conference papers along with several book chapters. He has 14 Ph.D. students and 80 M.S. thesis students finish under his supervision. His research interests include parallel computation, simulation, computer graphics, and virtual reality. He is a Senior Member of ACM and the International Society for Computers and their Applications (ISCA).