Node-Disjoint Paths Problems in Directed Bijective Connection Graphs

Keiichi KANEKO*(a), Member

SUMMARY In this paper, we extend the notion of bijective connection graphs to introduce directed bijective connection graphs. We propose algorithms that solve the node-to-set node-disjoint paths problem and the node-to-node node-disjoint paths problem in a directed bijective connection graph. The time complexities of the algorithms are both $O(n^3)$, and the maximum path lengths are both $2n - 1$.

key words: multicomputer, interconnection network, parallel processing, hypercube, fault tolerance, performance evaluation

1. Introduction

As the performance of sequential processing approaches its limit, there have been many studies on parallel processing, and they have proposed several topologies for interconnection networks of parallel systems. The hypercube $Q_n$ is a popular topology that has been adopted in many parallel systems. It has many variants such as the twisted cube $TQ_n$, the crossed cube $CQ_n$, the Möbius cube $0-MQ_n$ and $1-MQ_n$, the locally twisted cube $LTQ_n$, the spined cube $SQ_n$, and the twisted crossed cube $TCQ_n$. Table 1 shows a comparison of the diameters of these topologies with the same order $2^n$ and the same degree $n$.

Bijective connection graphs were proposed by Fan and Jia, and they provide a family of cube-based topologies. That is, they cover the hypercube and almost all of its variants with the order $2^n$ and the degree $n$. Hence, it is very important to design algorithms that work in a bijective connection graph. In this paper, we propose an extended family of topologies, directed bijective connection graphs, that are obtained by introducing directions with edges of bijective connection graphs.

It is also important to solve the node-disjoint paths problems in a parallel system so that it can achieve the best performance in communication. For a source node $s$ and a destination node $d$ in an $n$-connected graph, the node-to-node node-disjoint paths problem is to construct $n$ paths between $s$ and $d$. For a source node $s$ and a set $D$ of $n$ destination nodes $d_i$ (1 $\leq$ $i$ $\leq$ $n$) in an $n$-connected graph, the node-to-set node-disjoint paths problem is to construct $n$ paths between $s$ and $d_i$ (1 $\leq$ $i$ $\leq$ $n$) that are node-disjoint except for $s$. With a hypercube and its variants, these node-disjoint paths problems have been solved [9]–[15]. We summarize the results in Table 2. In this paper, we focus on these two problems in directed bijective connection graphs, and propose algorithms that solve them.

The rest of this paper is structured as follows. In Sect. 2, we define the bijective connection graphs and the directed bijective connection graphs with two lemmas with respect to the properties of the directed bijective connection graphs. In Sects. 3 and 4, we give algorithms that solve the node-to-set disjoint paths problem and the node-to-node disjoint paths problem in a directed bijective connection graph, respectively. In Sect. 5, we prove the correctness of the algorithms, and estimate their time complexities and the maximum lengths of the paths constructed by them. In Sect. 6, we conclude this paper and give a future work.

2. Preliminaries

In this section, we define bijective connection graphs and directed bijective connection graphs, and prove two lemmas regarding properties of directed bijective connection graphs.
Fig. 1 Elements of $L_2$ (a) and $L_2'$ (b) and (c).

Now, we give a definition of bijective connection graphs.

**Definition 1**: A class of $n$-dimensional bijective connection graphs $L_n$ is recursively defined as follows. $L_0 = \{ \vec{B}_0 \}$ where $|V(\vec{B}_0)| = 1$ and $E(\vec{B}_0) = \emptyset$. Note that $\vec{B}_0$ consists of a single node. $\vec{B}_n \in L_n$ if and only if there are two node-disjoint bijective connection graphs $\vec{B}^{0}_{n-1}$ and $\vec{B}^{1}_{n-1}$ in $L_{n-1}$ such that $V(\vec{B}_n) = V(\vec{B}^{0}_{n-1}) \cup V(\vec{B}^{1}_{n-1})$, and $E(\vec{B}_n) = E(\vec{B}^{0}_{n-1}) \cup E(\vec{B}^{1}_{n-1}) \cup E_n$ where there is a bijection $\eta_n : V(\vec{B}^{0}_{n-1}) \to V(\vec{B}^{1}_{n-1})$ such that for any node $x$ in $\vec{B}^{0}_{n-1}$, $(x, \eta_n(x)) \in E_n$.

By associating a direction with each edge, we define directed bijective connection graphs.

**Definition 2**: A class of $n$-dimensional directed bijective connection graphs $\vec{L}_n$ is recursively defined as follows. $\vec{L}_0 = \{ \vec{B}_0 \}$ where $|V(\vec{B}_0)| = 1$ and $E(\vec{B}_0) = \emptyset$. Note that $\vec{B}_0$ also consists of a single node. $\vec{B}_n \in \vec{L}_n$ if and only if there are two node-disjoint directed bijective connection graphs $\vec{B}^{0}_{n-1}$ and $\vec{B}^{1}_{n-1}$ in $\vec{L}_{n-1}$ such that $V(\vec{B}_n) = V(\vec{B}^{0}_{n-1}) \cup V(\vec{B}^{1}_{n-1})$, and $E(\vec{B}_n) = E(\vec{B}^{0}_{n-1}) \cup E(\vec{B}^{1}_{n-1}) \cup E_n$ where there are two bijections $\xi_n$ and $\zeta_n$ such that for any node $x$ in $\vec{B}^{0}_{n-1}$, $(x, \xi_n(x)) \in E_n$ and for any node $y$ in $\vec{B}^{1}_{n-1}$, $(y, \zeta_n(y)) \in E_n$.

$L_n$ and $\vec{L}_n$ give different families of graphs. For example, $L_2$ is a singleton set whose element is shown in Fig. 1 (a) while $\vec{L}_2$ contains two elements shown in Fig. 1 (b) and (c).

In the rest of this paper, let $\vec{B}_n$ represent an arbitrary directed bijective connection graph in $\vec{L}_n$.

**Definition 3**: For a node $a$ in $\vec{B}_n$, let the neighbor node of $a$ with an edge of either $(a, \xi(a))$ or $(a, \zeta(a))$ be denoted by $\phi_i(a)$. Also, let the neighbor node of $a$ with an edge of either $(\xi^{-1}(a), a)$ or $(\zeta^{-1}(a), a)$ be denoted by $\psi_i(a)$.

Then, each node $a$ in $\vec{B}_n$ has $n$ neighbor nodes $\phi_i(a)$ $(1 \leq i \leq n)$ with $(a, \phi_i(a)) \in E(\vec{B}_n)$ and $n$ neighbor nodes $\psi_i(a)$ $(1 \leq i \leq n)$ with $(\psi_i(a), a) \in E(\vec{B}_n)$. Figure 2 shows these $2n$ neighbor nodes of the node $a$.

In this paper, a path is an alternating finite sequence of nodes and edges, $a_0, e_0, a_1, e_1, a_2, \ldots, a_{k-1}, e_{k-1}, a_k$ where $e_i = (a_i, a_{i+1})$ $(0 \leq i \leq k - 1)$. The length of a path is the number of edges included in it. We use the shorthand notation $a_0 \to a_1 \to \cdots \to a_k$ or $a_0 \leadsto a_k$ 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 the word ‘disjoint’ instead of the word ‘node-disjoint’ in the rest of this paper.

**Lemma 1**: For $\vec{B}_n$ that consists of two node-disjoint directed bijective graphs $\vec{B}^{0}_{n-1}$ and $\vec{B}^{1}_{n-1}$, assume a node $a$ is in $\vec{B}^{0}_{n-1}$. Then, we can construct $n$ paths from $\vec{B}^{0}_{n-1}$ to $a$: $P_i: \vec{B}^{0}_{n-1} \leadsto a$ $(1 \leq i \leq n)$ in $O(n)$ time such that the paths are disjoint except for $a$ and their lengths are at most 2.

**Proof**: For $a$, we can construct $n$ disjoint paths:

$$
\begin{align*}
P_i & : \psi_i(\psi_i(a)) \to \psi_i(a) \to a \\
& \quad (1 \leq i \leq n - 1) \\
& \psi_n(a) \to a \\
& \quad (i = n)
\end{align*}
$$

The length of the path $P_n$ is 1, and the lengths of the remaining $(n - 1)$ paths $P_i$ $(1 \leq i \leq n - 1)$ are all 2. It takes $O(1)$ time to construct a path. Hence, it takes $O(n)$ time in total.

Figure 3 shows the $n$ paths from $\vec{B}^{0}_{n-1}$ to $a$ $(\in V(\vec{B}^{1}_{n-1}))$ that are disjoint except for $a$.

**Lemma 2**: For two nodes $s, d$ $(\in V(\vec{B}_n))$, there is an algorithm $R$ that constructs a path $s \sim d$ of length at most $n$ in $O(n)$ time.

**Proof**: If $s = d$, Algorithm $R$ can construct the path of length 0. Hence, we assume that $s \neq d$. Then, Algorithm $R$ is given as follows.

**Step 1** If $n = 1$, then since $s$ and $d$ are adjacent in $\vec{B}_1$, select the edge $(s, d)$, and terminate.

**Step 2** If $s$ and $d$ are both in a sub graph $\vec{B}^{i}_{n-1}$ $(i \in [0, 1])$, then apply Algorithm $R$ in $\vec{B}^{i}_{n-1}$ recursively, and terminate.

**Step 3** Otherwise, if $s \in V(\vec{B}^{0}_{n-1})$ and $d \in V(\vec{B}^{1}_{n-1})$ $(i \in [0, 1])$, select the edge $(s, \phi_n(s))$ and apply Algorithm $R$.
3. Node-to-Set Disjoint Paths Algorithm

In $\vec{B}_1$, we can construct a path of length 1 from a source node to a destination node in $O(1)$ time. Therefore, in this section, we assume that $n \geq 2$, and propose an algorithm N2S that solves the node-to-set disjoint paths problem for a source node $s$ and a destination node set $D = \{d_1, d_2, \ldots, d_n\}$ in $\vec{B}_n$, which consists of $\vec{B}_0^n$ and $\vec{B}_1^n$.

Algorithm N2S is divided into two cases depending on the distribution of the source node $s$ and the destination nodes $d_1, d_2, \ldots, d_n$. Since we can exchange $\vec{B}_0^n$ and $\vec{B}_1^n$ without loss of generality, we assume that $s \in V(\vec{B}_0^n)$ for simplicity.

3.1 Case 1

In this case, we assume that $\vec{B}_0^n$ contains all of the destination nodes.

**Step 1** Apply Algorithm N2S in $\vec{B}_0^n$ recursively to construct $(n-1)$ paths from $s$ to $d_1, d_2, \ldots, d_{n-1}$ that are disjoint except for $s$ (Fig. 4).

**Step 2** If $d_n$ lies on one of the paths, say $s \leadsto d_h$, constructed in Step 1, discard the sub path $d_n \leadsto d_h$, and exchange the indices of $d_h$ and $d_n$ (Fig. 5).

**Step 3** Select the edges $(s, \phi_n(s))$ and $(\psi_n(d_n), d_n)$.

**Step 4** In $\vec{B}_1^n$, apply Algorithm R to construct a path $\phi_n(s) \leadsto \psi_n(d_n)$ (Fig. 6).

3.2 Case 2

In this case, we assume that some destination nodes are included in $\vec{B}_1^n$. Without loss of generality, we can assume $\{d_1, d_2, \ldots, d_k\} \subset V(\vec{B}_0^n)$ and $\{d_{k+1}, d_{k+2}, \ldots, d_n\} \subset V(\vec{B}_1^n)$ (Fig. 7).

**Step 1** For each destination node $d_i$ ($k+1 \leq i \leq n$) in $\vec{B}_1^n$, consider the $n$ paths from $\vec{B}_0^n$ to $d_i$ from Lemma 1, and select one of them $d_i' \in V(\vec{B}_0^n)$ that is disjoint from other destination nodes and the paths to them. Assume that the path $\psi_n(d_i') \leadsto d_i$ is always selected if it is available (Fig. 8).

**Step 2** Select the edge $s \rightarrow \phi_n(s)$. 

---

Fig. 4 After Case 1, Step 1 of Algorithm N2S.

Fig. 5 After Case 1, Step 2 of Algorithm N2S.

Fig. 6 After Case 1, Step 4 of Algorithm N2S.

Fig. 7 Case 2 ($D \not\subseteq \vec{B}_0^n$) of Algorithm N2S.

Fig. 8 After Case 2, Step 1 of Algorithm N2S.
Step 3 In \( \tilde{B}_{n-1} \), apply Algorithm R to construct a path \( \phi_n(s) \leadsto d_n \) (Fig. 9).

Step 4 If the path \( \phi_n(s) \leadsto d_n \) does not contain any node on the other paths constructed in Step 1, discard the path \( d_n' \in V(\tilde{B}_{n-1}) \leadsto d_n \). Otherwise, let \( d_h' \) be one such node that is closest to \( \phi_n(s) \) and on the path \( d_h' \in V(\tilde{B}_{n-1}) \leadsto d_n \). Discard the sub paths \( d_h' \in V(\tilde{B}_{n-1}) \leadsto d_n' \) and \( d_h' \leadsto d_n \), and exchange the indices of \( d_h' \) and \( d_n' \) as well as \( d_h' \) and \( d_n \) (Fig. 10).

Step 5 Apply Algorithm N2S in \( \tilde{B}_{n-1} \) recursively to construct \( (n-1) \) paths from \( s \) to \( d_1, d_2, \ldots, d_k, d_{k+1}, \ldots, d_{n-1} \) that are disjoint except for \( s \) (Fig. 11).

For a source node \( s \) and a destination node \( d \) in \( \tilde{B}_n \), by applying Algorithm N2S to \( s \) and \( D = \{ \psi_1(d), \psi_2(d), \ldots, \psi_n(d) \} \), we can construct \( n \) paths of lengths at most \( 2n \) between \( s \) and \( d \) that are disjoint except for \( s \) and \( d \) in \( O(n^4) \) time. However, in the next section, we propose an algorithm N2N that can construct \( n \) paths between \( s \) and \( d \) in \( O(n^4) \) time such that the paths are disjoint except for \( s \) and \( d \) and their lengths are at most \( 2n-1 \).

4. Node-to-Node Disjoint Paths Algorithm

Again, in \( \tilde{B}_1 \), we can construct a path of length \( 1 \) from a source node to a destination node in \( O(1) \) time. Therefore, in this section, we assume that \( n \geq 2 \), and propose an algorithm N2N that solves the node-to-node disjoint paths problem for a source node \( s \) and a destination node \( d \) in \( \tilde{B}_n \), which consists of \( \tilde{B}_{n-1} \) and \( \tilde{B}_1 \).

Algorithm N2N is divided into three cases depending on the distribution of the source node \( s \) and the destination node \( d \). Since we can exchange \( \tilde{B}_{n-1} \) and \( \tilde{B}_1 \) without loss of generality, we assume that \( s \in V(\tilde{B}_{n-1}) \) for simplicity.

4.1 Case 1

In this case, we assume that \( \tilde{B}_{n-1} \) contains the destination node.

Step 1 Apply Algorithm N2N in \( \tilde{B}_{n-1} \) recursively to construct \( (n-1) \) paths from \( s \) to \( d \) that are disjoint except for \( s \) and \( d \).

Step 2 Select the edges \( (s, \phi_n(s)) \) and \( (\psi_n(d), d) \).

Step 3 In \( \tilde{B}_{n-1} \), apply Algorithm R to construct a path \( \phi_n(s) \leadsto \psi_n(d) \) (Fig. 12).

4.2 Case 2

In this case, we assume that the destination node is included in \( \tilde{B}_{n-1} \) and there is not an edge from the source node to the destination node.

Step 1 Select the edge \( (s, \phi_n(s)) \).

Step 2 Apply Algorithm R to construct the path \( \phi_n(s) \leadsto d \).

Let \( \psi_j(d) \) be the neighbor node of \( d \) that is included in the path (Fig. 13).

Step 3 Select \( (n-2) \) paths \( \psi_i(\psi_j(d)) \rightarrow \psi_i(d) \rightarrow d \) \((1 \leq i \neq j \leq n-1)\) and the edge \( (\psi_n(d), d) \) (Fig. 14).

Step 4 In \( \tilde{B}_{n-1} \), apply Algorithm N2S to construct \( (n-1) \) paths from \( s \) to \( \psi_i(\psi_j(d)) \) \((1 \leq i \neq j \leq n-1)\) and \( \psi_n(d) \) that are disjoint except for \( s \) (Fig. 15).
4.3 Case 3

In this case, we assume that the destination node is included in \( B_{n-1} \) and there is an edge from the source node to the destination node.

Step 1 From Lemma 1, select \((n-1)\) paths \( \psi_n(\psi_i(d)) \rightarrow \psi_i(d) \rightarrow d \) (1 \( \leq i \leq n-1 \)) and the edge \((s = \psi_n(d), d)\) (Fig. 16).

Step 2 In \( B_{n-1} \), apply Algorithm N2S to construct \((n-1)\) paths from \( s \) to \( \psi_n(\psi_i(d)) \) (1 \( \leq i \leq n-1 \)) that are disjoint except for \( s \) (Fig. 17).

5. Correctness, Time Complexities and Maximum Path Lengths

In this section, we prove correctness of Algorithms N2S and N2N, and estimate their time complexities and the maximum lengths of the paths constructed by them. For discussion, let \( T_0(n) \) and \( L_0(n) \) represent the time complexity of Algorithm N2S and the maximum length of the paths constructed by it, respectively. Also, let \( T_1(n) \) and \( L_1(n) \) represent the time complexity of Algorithm N2N and the maximum length of the paths constructed by it, respectively.

Lemma 3: In Case 1, Algorithm N2S can construct \( n \) paths from \( s \) to \( d_1, d_2, \ldots, d_n \) that are disjoint except for \( s \) in \( T_0(n) = T_0(n-1) + O(L_0(n) \times (n-1)) \) time. The maximum path length is \( L_0(n) = \max\{L_0(n-1), n + 1\} \).

(Proof) Step 1 takes \( T_0(n-1) \) time to construct \((n-1)\) paths of lengths at most \( L_0(n-1) \). The paths are disjoint except for \( s \) from the induction hypothesis. Step 2 takes \( O(L_0(n-1) \times (n-1)) \) time to check if \( d_n \) lies on one of the paths constructed in Step 1. If it is on the path \( s \leadsto d_n \), it takes \( O(1) \) time to discard the sub path and exchange the indices of \( d_n \) and \( d_m \). Step 3 takes \( O(1) \) time to select the two edges. From Lemma 2, Step 4 takes \( O(n) \) time to construct the path \( \phi_n(s) \leadsto \psi_n(d_n) \) of length at most \( n-1 \). The path \( s \leadsto \phi_n(s) \leadsto \psi_n(d_n) \leadsto d_n \) is disjoint from other paths constructed in Step 1 except for \( s \) since it is outside of \( B_{n-1} \) except for \( s \) and \( d_n \), and its length is \( (n-1) + 2 = n + 1 \). Consequently, Algorithm N2S takes \( T_0(n) = T_0(n-1) + O(L_0(n-1) \times (n-1)) \) time to construct \( n \) paths from \( s \) to \( d_1, d_2, \ldots, d_n \) that are disjoint except for \( s \). The maximum path length is \( L_0(n) = \max\{L_0(n-1), n + 1\} \).

Lemma 4: In Case 2, Algorithm N2S can construct \( n \) paths from \( s \) to \( d_1, d_2, \ldots, d_n \) that are disjoint except for \( s \) in \( T_0(n) = O(n^3) \) time. The maximum path length is \( L_0(n) = 2n - 1 \).
(Proof) In Step 1, it takes $O(n^2)$ time to select the path $d'_i \leadsto d_i$ for each $d_i$. Even if a path $d'_i \leadsto d_j$ $(i \neq j)$ of length 2 blocks two of the $n$ paths given by Lemma 1, it means that $\psi_n(d_j) \in D$ since the path $\psi_n(d_j) \leadsto d_j$ is not selected for $d'_i$, and the path $d'_i \leadsto d_j$ blocks only one path given by Lemma 1 essentially. Hence, we can always select a path for $d_i$ that is disjoint from other destination nodes and the paths to them. To select $(n-k)$ paths for $d_i$ $(k+1 \leq i \leq n)$, it takes $O(n^3)$ time. The path lengths are at most 2. Step 2 takes $O(1)$ time to select the edge. From Lemma 2, Step 3 takes $O(n)$ time to construct the path $\psi_n(s) \leadsto d_s$ of length at most $n-1$. In Step 4, it takes $O(n^2)$ time to check if the path $\phi_n(s) \leadsto d_s$ contains some nodes on the other paths constructed in Step 1. The remaining process in Step 4 takes at most $O(n)$ time. The sub paths are all disjoint. Step 5 takes $T_0(n-1)$ time to construct $(n-1)$ paths of length at most $L_0(n-1)$. From the induction hypothesis, the paths are disjoint except for $s$. Consequently, Algorithm N2S takes $T_0(n) = T_0(n-1) + O(n^3)$ time to construct $n$ paths from $s$ to $d_1, d_2, \ldots, d_n$ that are disjoint except for $s$. The maximum path length is $L_0(n) = \max[L_0(n-1) + 2, n]$. Solving these two equations with $L(1) = 1$, we have $T_0(n) = O(n^3)$ time and $L_0(n) = 2n - 1$. \hfill \Box

**Theorem 1:** Algorithm N2S can construct $n$ paths from $s$ to $d_1, d_2, \ldots, d_n$ that are disjoint except for $s$ in $T_0(n) = O(n^3)$ time. The maximum path length is $L_0(n) = 2n - 1$. (Proof) From Lemmas 3 and 4, the $n$ paths $s \leadsto d_i$ $(1 \leq i \leq n)$ are disjoint except for $s$, and we have $T_0(n) = O(n^3)$ time and $L_0(n) = 2n - 1$. \hfill \Box

**Lemma 5:** In Case 1, Algorithm N2N can construct $n$ paths from $s$ to $d$ that are disjoint except for $s$ and $d$ in $T_1(n) = T_1(n-1) + O(n)$ time. The maximum path length is $L_1(n) = \max[L_1(n-1), n+1]$. (Proof) Step 1 takes $T_1(n-1)$ time to construct $(n-1)$ paths of length at most $L_1(n-1)$. The paths are disjoint except for $s$ and $d$ from the induction hypothesis. Step 2 takes $O(1)$ time to select two edges. From Lemma 2, Step 3 takes $O(n)$ time to construct the path $\phi_n(s) \leadsto \psi_n(d)$ of length at most $n-1$. The path $s \leadsto \phi_n(s) \leadsto \psi_n(d) \leadsto d$ is disjoint from other paths constructed in Step 1 except for $s$ and $d$ since it is outside of $B^0_{n-1}$ except for $s$ and $d$, and its length is $(n-1) + 2 = n + 1$. Consequently, Algorithm N2N takes $T_1(n) = T_1(n-1) + O(n)$ time to construct $n$ paths from $s$ to $d$ that are disjoint except for $s$ and $d$. The maximum path length is $L_1(n) = \max[L_1(n-1), n+1]$. \hfill \Box

**Lemma 6:** In Case 2, Algorithm N2N can construct $n$ paths from $s$ to $d$ that are disjoint except for $s$ and $d$ in $T_1(n) = O(n^4)$ time. The maximum path length is $L_1(n) = 2n - 1$. (Proof) Step 1 takes $O(1)$ time to select the edge. From Lemma 2, Step 2 takes $O(n)$ time to construct the path $\phi_n(s) \leadsto \psi_n(d)$ of length at most $n-1$. Step 3 takes $O(n)$ time to construct the $(n-2)$ paths $\psi_n(\psi_1(d)) \leadsto d$ $(1 \leq i \neq j \leq n-1)$ of length 2 and to select the edge $\phi_n(d) \leadsto d$. These sub paths and the edge are all disjoint except for $d$. From Theorem 1, Step 4 takes $O(n^3)$ time to construct $(n-1)$ paths of length at most $2n - 3$ that are disjoint except for $s$. Consequently, Algorithm N2N takes $T_1(n) = O(n^3)$ time to construct $n$ paths from $s$ to $d$ that are disjoint except for $s$ and $d$. The maximum path length is $L_1(n) = \max[2n - 3 + 2, (n-1) + 1] = 2n - 1$. \hfill \Box

**Lemma 7:** In Case 3, Algorithm N2N can construct $n$ paths from $s$ to $d$ that are disjoint except for $s$ and $d$ in $T_1(n) = O(n^3)$ time. The maximum path length is $L_1(n) = 2n - 1$. (Proof) Step 1 takes $O(n)$ time to construct $n$ paths of lengths at most 2 from the nodes $\psi_n(\psi_1(d)), \psi_n(\psi_2(d)), \ldots, \psi_n(\psi_{n-1}(d)), \psi_n(d)$. The paths are disjoint except for $d$. From Theorem 1, Step 2 takes $O(n^3)$ time to construct $(n-1)$ paths of length at most $2n - 3$ that are disjoint except for $s$. Consequently, Algorithm N2N takes $T_1(n) = O(n^3)$ time to construct $n$ paths from $s$ to $d$ that are disjoint except for $s$ and $d$. The maximum path length is $L_1(n) = 2n - 3 + 2 = 2n - 1$. \hfill \Box

**Theorem 2:** Algorithm N2N can construct $n$ paths from $s$ to $d$ that are disjoint except for $s$ and $d$ in $T_1(n) = O(n^4)$ time. The maximum path length is $L_1(n) = 2n - 1$. (Proof) From Lemmas 5, 6, and 7, the $n$ paths from $s$ to $d$ are disjoint except for $s$ and $d$, and we have $T_1(n) = O(n^4)$ time and $L_1(n) = 2n - 1$. \hfill \Box

### 6. Computer Experiments

In this section, we evaluate the average performance of Algorithms N2S and N2N. For this purpose, we implemented the algorithms and then applied them to $n$-dimensional locally twisted cubes $LTQ_n$ $(2 \leq n \leq 31)$ because $LTQ_n$ does not have any disjoint path algorithms.

First, we conducted a computer experiment regarding Algorithm N2S in $LTQ_n$ by repeating the following procedure 1,000,000 times for each $n$:

1. Select a source node $s$ randomly.
2. Select $n$ distinct destination nodes $d_1, d_2, \ldots, d_n$ randomly. Note that it is allowed to select $s$ as one of the destination nodes.
3. Apply Algorithm N2S, and measure the path lengths and the execution time.

Figure 18 shows the maximum and average path lengths obtained in the experiment. We can see that the maximum length of the obtained paths is almost $n - 2$. The result is much smaller than the theoretical maximum path length $2n - 1$. In addition, we can see that the average path length is almost $n/2$.

Figure 19 shows the average execution time. We can see that the average time complexity of Algorithm N2S applied to $LTQ_n$ is almost $O(n^2)$. The result is much smaller than the theoretical time complexity $O(n^4)$.

Next, we also conducted a computer experiment regarding Algorithm N2N in $LTQ_n$ by repeating the following procedure 1,000,000 times for each $n$:
1. Select a source node $s$ randomly.
2. Select a destination node $d$ randomly. Note that it is allowed to select $s$ as the destination node.
3. Apply Algorithm N2N, and measure the path lengths and the execution time.

Figure 20 shows the maximum and average path lengths obtained in the experiment. We can see that the maximum length of the obtained paths is at most $n + 1$. The result is much smaller than the theoretical maximum path length $2n - 1$. In addition, we can see that the average path length is almost $n/2 + 1$.

Figure 21 shows the average execution time. We can see that the average time complexity of Algorithm N2N applied to $LT Q_n$ is also almost $O(n^2)$. The result is much smaller than the theoretical time complexity $O(n^3)$.

7. Conclusion and Future Work

In this paper, we introduced directed bijective connection graphs and proposed two algorithms that solve the node-to-set disjoint paths problem and the node-to-node disjoint paths problem in directed bijective connection graphs. We have proved the correctness of the algorithms and estimated that their time complexities are both $O(n^3)$, and the maximum path lengths are both $2n - 1$. We also have conducted computer experiments in $LT Q_n$ ($2 \leq n \leq 31$), and found the average time complexities of the algorithms are both $O(n^2)$. Also, we found that the theoretical maximum path lengths are hard to be attained, and the average path length of the node-to-set disjoint path algorithm is almost $n/2$ while that of the node-to-node disjoint path algorithm is $n/2 + 1$.

Future works include development of an algorithm that solves the set-to-set disjoint paths problem in directed bijective connection graphs.

Acknowledgments

The author would like to express special thanks to Dr. Bipin Indurkhya for proofreading the earlier version of this paper. He also appreciates the reviewers for their insightful comments based on their careful reading. 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. 17K00093.

References


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