PAPER *Special Section on Parallel/Distributed Computing and Networking*

An Algorithm for Node-to-Set Disjoint Paths Problem in Bi-Rotator Graphs

Keiichi KANEKO[†a)], ***Member***

SUMMARY An algorithm is described for solving the node-to-set disjoint paths problem in bi-rotator graphs, which are obtained by making each edge of a rotator graph bi-directional. The algorithm is of polynomial order of n for an n-bi-rotator graph. It is based on recursion and divided into three cases according to the distribution of destination nodes in the classes into which the nodes in a bi-rotator graph are categorized. We estimated that it obtains $2n-3$ disjoint paths with a time complexity of $O(n^5)$, that the sum of the path lengths is $O(n^3)$, and that the length of the maximum path is $O(n^2)$. Computer experiment showed that the average execution time was $O(n^{3.9})$ and, the average sum of the path lengths was $O(n^{3.0})$.

key words: *node-to-set disjoint paths problem, bi-rotator graphs, fault tolerance, parallel computation, routing algorithm*

1. Introduction

Ongoing research on parallel and distributed computation and on massively parallel processing have led to proposals for interconnection networks based on complicated topologies, such as meshes, tori, and hypercubes. Most of those new topologies are variants of the Cayley graph [1], [6] and have been the focus of many intensive research efforts [2]–[5], [7]–[11], [13], [15]. The rotator graph [6] is one such topology and is very promising because of its small diameter and small degree. However, a rotator graph is a directed graph that sometimes behaves harmfully when applied to actual problems.

A bi-rotator graph [12] is obtained by making each edge of a rotator graph bi-directional. A bi-rotator graph has a Hamilton cycle and is also pan-cyclic. Among the unresolved problems with bi-rotator graphs is the node-to-set disjoint paths problem: in k-connected graph $G = (V, E)$, for source node $\boldsymbol{s}$ and a set of k destination nodes, $D = \{\boldsymbol{d}_1, \boldsymbol{d}_2, \cdots, \boldsymbol{d}_k\}$ ($\boldsymbol{s} \notin D$), find k paths from $\boldsymbol{s}$ to $\boldsymbol{d}_i$ ($1 \le i \le k$) that are node-disjoint except for $\boldsymbol{s}$. This is an important issue in parallel and distributed computing [5], [8], [10], [14] as well as the node-to-node disjoint paths problem [9], [15].

In general, node-disjoint paths are obtained by using the maximum flow algorithm in polynomial order of the number of the nodes in the graph, $|V|$. There are $n!$ nodes in an n-bi-rotator graph. Hence, this approach is impractical. We thus developed an algorithm of polynomial order of n instead of $n!$ and proved its correctness and estimated its theoretical performance. We also conducted the computer experiment to evaluate its average performance.

The rest of this paper is structured as follows. Section 2 gives some definitions and properties. Section 3 describes our algorithm in detail and the proof of its correctness and the estimation of its complexities are given in Sect. 4. In Sect. 5, the computer experiment is conducted. We conclude and give future works in Sect. 6.

Manuscript received March 28, 2005.
Manuscript revised August 15, 2005.

[†]The author is with the Faculty of Engineering, Tokyo University of Agriculture and Technology, Koganei-shi, 184–8588 Japan.
a) E-mail: k1kaneko@cc.tuat.ac.jp

DOI: 10.1093/ietisy/e89–d.2.647

2. Preliminaries

In this section, we define a bi-rotator graph and a class and then describe a simple unicast routing algorithm called `route`.

Definition 1: For arbitrary permutation $\boldsymbol{u} = (u_1, u_2, \cdots, u_n)$ of n symbols $1, 2, \cdots, n$ and integer i ($2 \le i \le n$), we define positive and negative rotation operations $R_i^+(\boldsymbol{u})$ and $R_i^-(\boldsymbol{u})$ as follows:

$$R_i^+(\boldsymbol{u}) = (u_2, u_3, \cdots, u_i, u_1, u_{i+1}, u_{i+2}, \cdots, u_n),$$

$$R_i^-(\boldsymbol{u}) = (u_i, u_1, u_2, \cdots, u_{i-1}, u_{i+1}, u_{i+2}, \cdots, u_n).$$

Note that R_2^+ and R_2^- represent the same rotation operation and that there are $2n-3$ distinct rotation operations.

Definition 2: An n-bi-rotator graph, BR_n, has $n!$ nodes. Each node has a unique address that is a permutation of n symbols, $1, 2, \cdots, n$. A node with address $\boldsymbol{u} = (u_1, u_2, \cdots, u_n)$ is adjacent to the nodes with addresses that are elements of the set $\{R_i^+(\boldsymbol{u}), R_i^-(\boldsymbol{u}) \mid 2 \le i \le n\}$, and it is not adjacent to any other node. Let the neighbor node set of $\boldsymbol{u}$ be denoted by $N(\boldsymbol{u})$.

Table 1 compares an n-bi-rotator graph with an $n \times n$ torus, an n-dimensional hypercube, an (n, k)-de Bruijn graph, and an (n, k)-Kautz graph (T_n, Q_n, $B(n, k)$, and $K(n, k)$, respectively). The average diameter of an n-bi-rotator graph is unknown. In terms of the integration ratio, which is defined by the number of nodes/(degree × diameter), an n-bi-rotator graph is inferior to an (n, k)-de Bruijn graph and an (n, k)-Kautz graph. However, it has recursive structure, as described below, and it can easily implement algorithms such as divide-and-conquer in parallel.

Figure 1 shows examples of 2-, 3-, and 4-bi-rotator graphs. Note that the addresses $(u_1, u_2, \cdots, u_n)$ are denoted by $u_1 u_2 \cdots u_n$ to save space.

Definition 3: In an n-bi-rotator graph, a sub graph induced by the nodes that have a common symbol k at the right-most

Table 1 Comparison of a bi-rotator graph with other graphs.

	No. of nodes	Degree	Diameter	Integration
BR_n	$n!$	$2n-3$	$n-1$	$\frac{n!}{(n-1)(2n-3)}$
T_n	n^2	4	n	$n/4$
Q_n	2^n	n	n	$2^n/n^2$
$B(n,k)$	n^k	n	k	n^{k-1}/k
$K(n,k)$	n^k+n^{k-1}	n	k	$\frac{n^{k-1}+n^{k-2}}{k}$

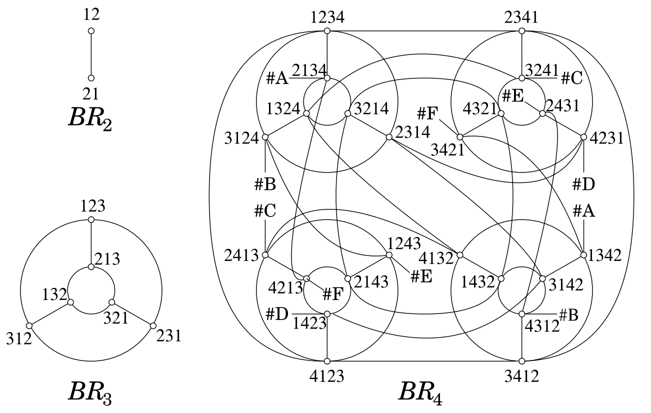

Fig. 1 Examples of 2-, 3-, and 4-bi-rotator graphs.

position in their addresses comprises an $(n-1)$-bi-rotator graph. A sub graph is denoted by $BR_{n-1}k$ by using the common symbol k.

Definition 4: For node $\boldsymbol{u}$ in an n-bi-rotator graph, BR_n, the class of $\boldsymbol{u}$ is the set of nodes obtained by applying $R_n^+(\cdot)$ to $\boldsymbol{u}$ repeatedly. We denote the class of $\boldsymbol{u}$ as $C(\boldsymbol{u})$.

The following properties hold for the classes in BR_n.

1. Each node belongs to exactly one class.
2. Each class consists of n nodes, which comprise a ring structure.
3. In each sub bi-rotator graph, exactly one node belongs to each class.
4. Each node has two neighbor nodes that belong to the same class as the node itself. The remaining $2n-5$ neighbor nodes each belong to different classes.

For instance, a 4-bi-rotator graph, BR_4, has six classes, each of which consists of four nodes:

$$C_1 = \{(1,2,3,4),(2,3,4,1),(3,4,1,2),(4,1,2,3)\}$$
$$C_2 = \{(1,3,2,4),(3,2,4,1),(2,4,1,3),(4,1,3,2)\}$$
$$C_3 = \{(2,1,3,4),(1,3,4,2),(3,4,2,1),(4,2,1,3)\}$$
$$C_4 = \{(2,3,1,4),(3,1,4,2),(1,4,2,3),(4,2,3,1)\}$$
$$C_5 = \{(3,1,2,4),(1,2,4,3),(2,4,3,1),(4,3,1,2)\}$$
$$C_6 = \{(3,2,1,4),(2,1,4,3),(1,4,3,2),(4,3,2,1)\}$$

Moreover, if we traverse the classes from the beginning, we find that they comprise a ring structure.

Definition 5: In a bi-rotator graph, a class path is a sub path that is part of a ring structure formed by a class.

The following lemma holds for classes.

Lemma 1: If $n \geq 4$, for two different nodes $\boldsymbol{u}$ and $\boldsymbol{v}$ that belong to a class in BR_n, there is a node among the neighbor nodes of $\boldsymbol{u}$ that belongs to a different class than those to which the neighbor nodes of $\boldsymbol{v}$ belong.

(Proof) Let $\boldsymbol{u} = (u_1, u_2, \cdots, u_n)$. Then, from $\boldsymbol{v} \in C(\boldsymbol{u})$ we can denote $\boldsymbol{v} = (u_k, u_{k+1}, \cdots, u_n, u_1, \cdots, u_{k-1})$ for some k. First we assume that $k \neq n-1$. In this case, take $\boldsymbol{w} = (u_{n-1}, u_1, u_2, \cdots, u_{n-2}, u_n)$. Then, $\boldsymbol{w}$ is a neighbor node of $\boldsymbol{u}$, and, from the fact that $n \geq 4$, it is not possible to insert u_{n-1} between u_n and u_1 with a single rotation operation for $\boldsymbol{v}$. Hence, there is no neighbor node of $\boldsymbol{v}$ that belongs to the same class as $\boldsymbol{w}$. Next, we assume that $k = n-1$. In this case, take $\boldsymbol{w} = (u_2, u_3, \cdots, u_{n-1}, u_1, u_n)$. Then, $\boldsymbol{w}$ is a neighbor node of $\boldsymbol{u}$, and, from the fact that $n \geq 4$, it is impossible to insert u_1 between u_{n-1} and u_n with a single rotation operation for $\boldsymbol{v}$. Hence, there is no neighbor node of $\boldsymbol{v}$ that belongs to the same class as $\boldsymbol{w}$. □

Because the shortest-path routing algorithm for an n-bi-rotator graph in polynomial-order time of n is still unknown, we use a simple unicast routing algorithm called `route`[12] that generates a path of length $O(n)$ in $O(n^2)$ time complexity. The two paths `route(`$\boldsymbol{s}$`, `$\boldsymbol{t}$`)` and `route(`$\boldsymbol{t}$`, `$\boldsymbol{s}$`)` are internally disjoint between $\boldsymbol{s}$ and $\boldsymbol{t}$[12].

In our algorithm, for four nodes $\boldsymbol{s}$, $\boldsymbol{t}$, $\boldsymbol{u}$, and $\boldsymbol{v}$ in BR_n it is necessary to construct a path from $\boldsymbol{s}$ to $\boldsymbol{t}$ without including nodes $\boldsymbol{u}$ and $\boldsymbol{v}$. If $n = 3$, there are three internally disjoint paths between $\boldsymbol{s}$ and $\boldsymbol{t}$ if they are not adjacent, and the problem is trivial. Therefore, we give an extended routing algorithm for $n \geq 4$:

Case 1 If $\exists h$ s. t. $\boldsymbol{s}, \boldsymbol{t}, \boldsymbol{u}, \boldsymbol{v} \in BR_{n-1}h$, apply the algorithm recursively inside $BR_{n-1}h$ and terminate.

Case 2 If $\exists h$ s. t. $\boldsymbol{s}, \boldsymbol{t} \in BR_{n-1}h$, $\boldsymbol{v} \notin BR_{n-1}h$, construct two internally disjoint paths between $\boldsymbol{s}$ and $\boldsymbol{t}$ inside $BR_{n-1}h$ and select one that does not include $\boldsymbol{u}$ and terminate.

Case 3 If $\exists h, l$ s. t. $h \neq l$, $\boldsymbol{s}, \boldsymbol{u}, \boldsymbol{v} \in BR_{n-1}h$, $\boldsymbol{t} \in BR_{n-1}l$, select a class path from $\boldsymbol{s}$ to $BR_{n-1}l \cap C(\boldsymbol{s})$ and select a path from $BR_{n-1}l \cap C(\boldsymbol{s})$ to $\boldsymbol{t}$ inside $BR_{n-1}l$ and terminate.

Case 4 If $\exists h, l$ s. t. $h \neq l$, $\boldsymbol{s} \in BR_{n-1}h$, $\boldsymbol{t} \in BR_{n-1}l$, $\boldsymbol{u} \notin BR_{n-1}h$, $\boldsymbol{v} \notin BR_{n-1}l$, select $\boldsymbol{x}$ so that $\boldsymbol{x} \in \{\boldsymbol{s}\} \cup N(\boldsymbol{s})$ s. t. $\boldsymbol{x} \notin C(\boldsymbol{u}) \cup C(\boldsymbol{v})$ holds. If $\boldsymbol{x} \neq \boldsymbol{s}$, select edge $(\boldsymbol{s}, \boldsymbol{x})$. Construct a class path from $\boldsymbol{x}$ to $BR_{n-1}l \cap C(\boldsymbol{x})$. In addition, construct two internally disjoint paths between $BR_{n-1}l \cap C(\boldsymbol{x})$ and $\boldsymbol{t}$ and select one that does not include $\boldsymbol{u}$ and terminate.

3. Algorithm

3.1 Classification

In a BR_3, the problem is trivial. Hence, we assume that $n \geq 4$. Taking advantage of the symmetric property of a BR_n, we fix the source node to $\boldsymbol{s} = (1, 2, \cdots, n)$. Let the destination node set be $D = \{\boldsymbol{d}_1, \boldsymbol{d}_2, \cdots, \boldsymbol{d}_{2n-3}\}$, and the set of classes to which the destination nodes belong be $C = \{C_1, C_2, \cdots, C_k\}$. Now let us consider the following cases:

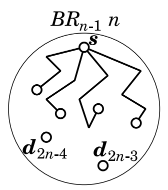

Fig. 2 Paths obtained after Step 1 of Procedure 1.

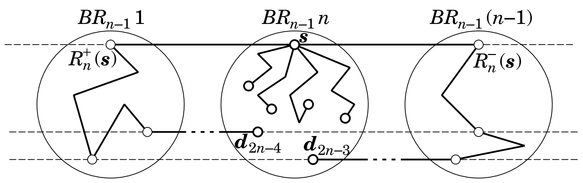

Fig. 3 Paths obtained after Step 2 of Procedure 1.

Case 1 All destination nodes belong to the same sub bi-rotator graph as the source node ($D \subset BR_{n-1}n$).

Case 2 Exactly one destination node is not included in the sub bi-rotator graph to which the source node belongs ($|D - BR_{n-1}n| = 1$).

Case 3 More than one destination node is not included in the sub bi-rotator graph to which the source node belongs ($|D - BR_{n-1}n| \geq 2$).

3.2 Procedure 1

Procedure 1 is used for Case 1 where $D \subset BR_{n-1}n$. It constructs $2n-3$ paths from source node s to destination node set $D = \{d_1, d_2, \cdots, d_{2n-3}\}$; the paths are node-disjoint except for the source node.

Step 1 Apply the algorithm recursively inside $BR_{n-1}n$ and obtain $2n-5$ paths from s to $D - \{d_{2n-4}, d_{2n-3}\}$ that are node-disjoint except for s. If one of these paths, say the path from s to d_m, contains d_{2n-4}, then discard the sub path from d_{2n-4} to d_m and exchange the indices of d_{2n-4} and d_m. For d_{2n-3}, perform a similar operation. After this step, we obtain paths shown in Fig. 2.

Step 2 Select edges $(s, R_n^+(s))$ and $(s, R_n^-(s))$. Then, construct class paths from destination nodes d_{2n-4} and d_{2n-3} to $C(d_{2n-4}) \cap BR_{n-1}1$ and $C(d_{2n-3}) \cap BR_{n-1}(n-1)$, respectively, without including the nodes $C(d_{2n-4}) \cap BR_{n-1}(n-1)$ and $C(d_{2n-3}) \cap BR_{n-1}1$. Apply `route` inside sub graphs $BR_{n-1}1$ and $BR_{n-1}(n-1)$, and construct paths from $C(d_{2n-4}) \cap BR_{n-1}1$ and $C(d_{2n-3}) \cap BR_{n-1}(n-1)$ to $R_n^+(s)$ and $R_n^-(s)$, respectively. After this step, we obtained the paths shown in Fig. 3. The horizontal dashed lines represent class ring structures. We assume that the both ends of each dashed line are connected.

3.3 Procedure 2

Procedure 2 is used for Case 2 where $|D - BR_{n-1}n| = 1$. It constructs $2n-3$ paths from source node s to destination node set $D = \{d_1, d_2, \cdots, d_{2n-3}\}$; the paths are node-disjoint except for the source node.

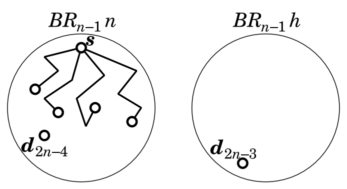

Fig. 4 Paths obtained after Step 1 of Procedure 2.

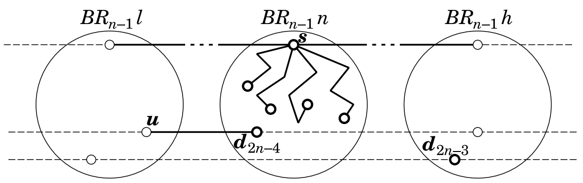

Fig. 5 Paths obtained after Step 2 of Procedure 2.

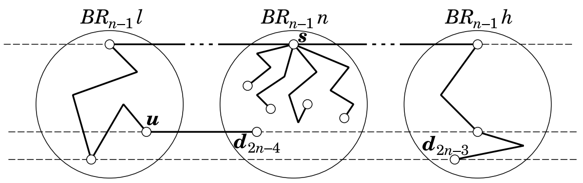

Fig. 6 Paths obtained after Step 3 of Procedure 2.

Step 1 We can assume that destination node d_{2n-3} is outside BR_{n-1} without loss of generality. We apply our algorithm recursively inside $BR_{n-1}n$ to obtain $2n-5$ paths from s to $D - \{d_{2n-4}, d_{2n-3}\}$ that are node-disjoint except for s. If one of these paths, say the path from s to d_m, contains d_{2n-4}, then discard the sub path from d_{2n-4} to d_m and exchange the indices of d_{2n-4} and d_m. After this step, we obtain the paths shown in Fig. 4.

Step 2 For destination node d_{2n-4}, apply either $R_n^+(\cdot)$ or $R_n^-(\cdot)$ to select edge (d_{2n-3}, u) such that u belongs to sub graph $BR_{n-1}l$ that is different from sub graph $BR_{n-1}h$ to which d_{2n-3} belongs. Construct two class paths from s to $C(s) \cap BR_{n-1}h$ and $C(s) \cap BR_{n-1}l$ that are node-disjoint except for s. After this step, we obtain the paths shown in Fig. 5.

Step 3 Inside sub graphs $BR_{n-1}h$ and $BR_{n-1}l$, apply `route` to construct paths from $C(s) \cap BR_{n-1}h$ and $C(s) \cap BR_{n-1}l$ to d_{2n-3} and u, respectively, and terminate. After this step, we obtain the paths shown in Fig. 6.

3.4 Procedure 3

Procedure 3 is used for Case 3 where $|D - BR_{n-1}n| \geq 2$. It constructs $2n-3$ paths from source node s to destination node set $D = \{d_1, d_2, \cdots, d_{2n-3}\}$; the paths are node-disjoint except for the source node.

Step 1 Let D_1 be the set of destination nodes that are

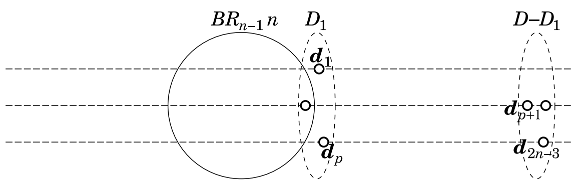

Fig. 7 Classification of destination nodes.

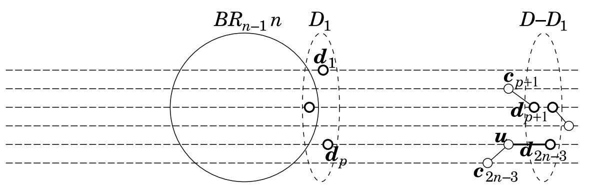

Fig. 8 Selection of neighbor nodes of $D - D_1$.

reached first by repeating rotation operation $R_n^+(\cdot)$ zero or more times from each node $C_i \cap BR_{n-1}n$. Additionally, if $C(\boldsymbol{s}) \in C$, then let D_1 include the destination node that is reached first by repeating $R_n^-(\cdot)$ one or more times from $\boldsymbol{s}$. Without loss of generality, we can assume that $D_1 = \{\boldsymbol{d}_1, \boldsymbol{d}_2, \cdots, \boldsymbol{d}_p\}$, and $D - D_1 = \{\boldsymbol{d}_{p+1}, \boldsymbol{d}_{p+2}, \cdots, \boldsymbol{d}_{2n-3}\}$. Figure 7 shows the classification of destination nodes.

Step 2 For each destination node $\boldsymbol{d}_i$ in $D - D_1$, find its neighbor node $\boldsymbol{c}_i$ that satisfies the following conditions in a greedy manner:

- $C(\boldsymbol{c}_i) \notin C$,
- $C(\boldsymbol{c}_i) \neq C(\boldsymbol{c}_j)$, if $i \neq j$.

From the fact that $|D-D_1| \leq 2n-4$ and Property 4 concerning classes, the neighbor node that satisfies these conditions can be selected except for the final destination node, say $\boldsymbol{d}_{2n-3}$. If all the neighbor nodes of $\boldsymbol{d}_{2n-3}$ belong to the classes of other destination nodes or the nodes $\boldsymbol{c}_i$, then perform the following process. If there exists a node in $\{R_n^+(\boldsymbol{d}_{2n-3}), R_n^-(\boldsymbol{d}_{2n-3})\} - BR_{n-1}n$ that is not a destination node, select the node, say $\boldsymbol{u}$, and edge $(\boldsymbol{d}_{2n-3}, \boldsymbol{u})$. Among the neighbor nodes of $\boldsymbol{u}$, select one node $\boldsymbol{c}_{2n-3}$ that does not belong to the class of any other destination node or any other $\boldsymbol{c}_i$. Lemma 1 ensures the existence of such a node. Figure 8 illustrates this case. Otherwise, if either $R_n^+(\boldsymbol{d}_{2n-3})$ or $R_n^-(\boldsymbol{d}_{2n-3})$ is a destination node that does not belong to D_1, then, for the destination node, say $\boldsymbol{d}_j$, release the previously selected node $\boldsymbol{c}_j$ and, from among the neighbor nodes of $\boldsymbol{d}_j$, newly select node $\boldsymbol{c}_j$ that does not belong to the class of any other destination node nor any other $\boldsymbol{c}_i$. Lemma 1 ensures the existence of such a node. Through this operation, it is possible to select the neighbor node $\boldsymbol{c}_{2n-3}$ from among the neighbor nodes of $\boldsymbol{d}_{2n-3}$ that belongs to the same class as the neighbor node of $\boldsymbol{d}_j$ that was previously released. If neither of the above conditions is satisfied, it induces that $R_n^+(\boldsymbol{d}_{2n-3}) \in BR_{n-1}n$ and $\boldsymbol{d}_j = R_n^-(\boldsymbol{d}_{2n-3}) \in D_1$. Then, remove $\boldsymbol{d}_j$ from D_1 and add $\boldsymbol{d}_{2n-3}$ to D_1. Moreover, for $\boldsymbol{d}_j$, select its neighbor node $\boldsymbol{c}_j$ that does not belong to the class of any other destination node or any other $\boldsymbol{c}_i$. Lemma 1 ensures the existence of such a node.

Step 3 For each destination node $\boldsymbol{d}_i$ in D_1, construct a class path from $C(\boldsymbol{d}_i) \cap BR_{n-1}n$ to $\boldsymbol{d}_i$ so that the path does not include any other destination nodes. Next, for each $\boldsymbol{c}_i$, construct a class path from the node in $C(\boldsymbol{c}_i) \cap BR_{n-1}n$ to $\boldsymbol{c}_i$. In these path constructions, if $\boldsymbol{d}_i$ is reachable with-

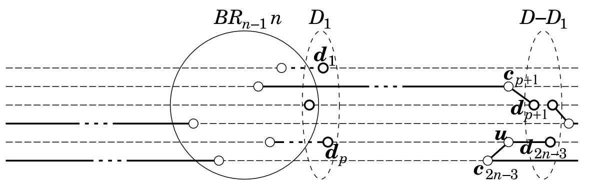

Fig. 9 Selection of class paths and edges.

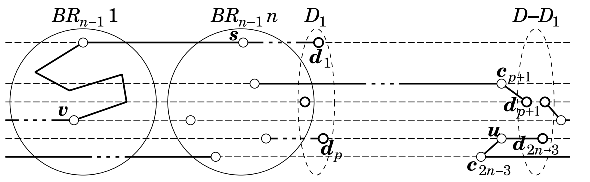

Fig. 10 Construction of a path.

out including other destinations by repetition of both of $R_n^+(\cdot)$ and $R_n^-(\cdot)$, select the shorter path. Select an edge between each $\boldsymbol{c}_i$ and corresponding $\boldsymbol{d}_i$. Figure 9 illustrates the selection of class paths and edges.

Step 4 If a path from $\boldsymbol{s}$ is not yet constructed, proceed to Step 5. Otherwise, if two paths from $\boldsymbol{s}$ are constructed, proceed to Step 7. In the rest of this step, we assume that there is exactly one path from $\boldsymbol{s}$ and that the path includes $R_n^-(\boldsymbol{s})$ without loss of generality. Select the edge between $\boldsymbol{s}$ and $R_n^+(\boldsymbol{s})$. If there is a node in $BR_{n-1}1$ that is included in a path already constructed, then construct a path from $R_n^+(\boldsymbol{s})$ to one of such nodes $\boldsymbol{v}$ without including other such nodes. Discard the sub path of the already constructed path from the node inside $BR_{n-1}n$ to $\boldsymbol{v}$. Figure 10 illustrates the construction of such a path. If there is no node in $BR_{n-1}1$ that is included in the path already constructed, for the nodes on the already constructed paths which have a length of one or more, consider the set of nodes in $BR_{n-1}1$ that belong to the same class as them but do not belong to the same class as the nodes on the path that include $\boldsymbol{s}$ as its terminal. If this set is not empty, construct a path from $R_n^+(\boldsymbol{s})$ to one node in the set, say $\boldsymbol{v}$, so that the path does not include any other such nodes. Let $\boldsymbol{w}$ be the first node on the already constructed path that is reachable by repeating either $R_n^+(\cdot)$ or $R_n^-(\cdot)$ from $\boldsymbol{v}$ without passing through sub graph $BR_{n-1}n$. For the path that was constructed in Step 3 and includes $\boldsymbol{w}$, discard its sub path from the terminal node in $BR_{n-1}n$ to $\boldsymbol{w}$. Construct a path from $\boldsymbol{v}$ to $\boldsymbol{w}$ by repeating either $R_n^+(\cdot)$ or $R_n^-(\cdot)$ without passing through sub graph $BR_{n-1}n$ and without

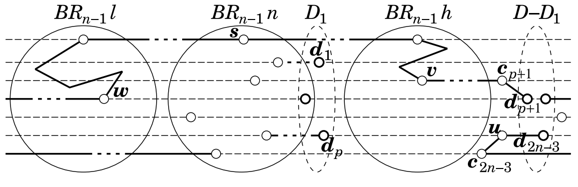

Fig. 11 Construction of two paths.

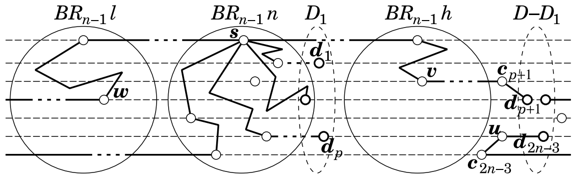

Fig. 12 Final results of recursive application of algorithm.

including other $\boldsymbol{c}_i$'s and $\boldsymbol{d}_i$'s. If neither of the above two conditions is satisfied, then $|D - BR_{n-1}n| = 2$ holds and these two destination nodes belong to the same class. First, discard the paths to these two destination nodes. Next, construct a class path from $\boldsymbol{s}$ to the nodes in the sub graphs to which these two destination nodes belong by repeating either $R_n^+(\cdot)$ or $R_n^-(\cdot)$. Moreover, construct paths to the destination nodes inside the sub graphs. Proceed to Step 7.

Step 5 Let $\boldsymbol{v}$ and $\boldsymbol{w}$ be two nodes that are, respectively, on already constructed distinct paths P_1 and P_2, and that are in different sub graphs $BR_{n-1}h$ and $BR_{n-1}l$ that are different from $BR_{n-1}n$. If these two nodes cannot be obtained, all the nodes on the already constructed paths belong to a single sub graph except for $BR_{n-1}n$. In this case, we can obtain these nodes by replacing one of the paths constructed in Step 3 with a length of one or more with the longer class path.

Step 6 Obtain class paths from $\boldsymbol{s}$ to nodes $\boldsymbol{t}_1$ and $\boldsymbol{t}_2$ in $BR_{n-1}h$ and $BR_{n-1}l$, respectively, by repeating $R_n^+(\cdot)$ and $R_n^-(\cdot)$. Additionally, construct paths from $\boldsymbol{t}_1$ and $\boldsymbol{t}_2$ to $\boldsymbol{v}$ and $\boldsymbol{w}$ while avoiding at most two nodes on P_2 and P_1 by applying the extended routing algorithm in $BR_{n-1}h$ and $BR_{n-1}l$, respectively. If these paths include nodes on already constructed paths other than $\boldsymbol{v}$ and $\boldsymbol{w}$, let the nearest nodes from $\boldsymbol{t}_1$ and $\boldsymbol{t}_2$ be $\boldsymbol{v}$ and $\boldsymbol{w}$, respectively. However, if two different nodes on a single path are obtained, keep the nearer node to the destination and ignore the other one. Discard sub paths from sub graph $BR_{n-1}n$ to $\boldsymbol{v}$ and $\boldsymbol{w}$. Figure 11 illustrates the construction of two paths.

Step 7 Apply the algorithm recursively inside $BR_{n-1}n$, and obtain $2n-5$ paths from $\boldsymbol{s}$ to the terminal nodes of paths other than $\boldsymbol{s}$ that are node-disjoint other than $\boldsymbol{s}$, and terminate. Figure 12 illustrates the final results of the recursive application of the algorithm.

4. Proof of Correctness and Estimation of Complexities

In this section, we give a proof of the correctness of our algorithm. We also give its time complexity and the sum and the maximum value of the path lengths obtained by our algorithm. We use the term 'disjoint' to express 'node-disjoint'.

Theorem 1: The paths generated by our algorithm are disjoint except for the source node $\boldsymbol{s}$. Let $T(n)$ be the time complexity of our algorithm, $L(n)$ be the sum of the path lengths, and $l(n)$ be the length of the maximum path generated by our algorithm for an n-bi-rotator graph. Then, $T(n) = O(n^5)$, $L(n) = O(n^3)$, and $l(n) = l(n-1) + \max\{\lfloor n/2 \rfloor + 2, n-2\}$.
(Proof) Based on induction on n, this theorem can be proved from the following lemmas. □

Lemma 2: The paths generated by Procedure 1 are disjoint except for the source node. The time complexity of Procedure 1 is $T(n-1) + L(n-1) \times O(n) + O(n^2)$. The sum of the path lengths is $L(n-1) + O(n)$, and the maximum path length is at most $2n-3$.
(Proof) The paths obtained in Step 1 are disjoint except for $\boldsymbol{s}$ from the hypothesis of induction. One of the two paths generated in Step 2 consists of the nodes in $BR_{n-1}1$ and the sub path of the class of $\boldsymbol{d}_{2n-4}$ that does not pass through $BR_{n-1}(n-1)$. Another path consists of the nodes in $BR_{n-1}(n-1)$ and the sub path of the class of $\boldsymbol{d}_{2n-3}$ that does not pass through $BR_{n-1}1$. Hence these two paths are disjoint with each other except for $\boldsymbol{s}$. Additionally, they contain the nodes outside $BR_{n-1}n$ except for these terminal nodes. Therefore, these two paths are also disjoint with the other paths generated in Step 1.

The time complexity of Step 1 is $T(n-1) + L(n-1) \times O(n)$. The sum of the path lengths generated in Step 1 is $L(n-1)$, and the length of the maximum path in Step 1 is at most $l(n-1)$. The time complexity of Step 2 is $O(n^2)$. The sum of the lengths of two paths obtained in Step 2 is $O(n)$, and the length of the maximum path in Step 2 is at most $2n-3$. Hence, the total time complexity of Procedure 1 is $T(n-1)+L(n-1)\times O(n)+O(n^2)$. The sum of the path lengths and the length of the maximum path obtained by Procedure 1 is $L(n-1) + O(n)$ and at most $2n-3$, respectively. □

Lemma 3: The paths generated by Procedure 2 are disjoint except for the source node. The time complexity of Procedure 2 is $T(n-1) + L(n-1) \times O(n) + O(n^2)$. The sum of the path lengths is $L(n-1) + O(n)$, and the length of the maximum path is at most $2n-3$.
(Proof) The paths obtained in Step 1 are disjoint except for $\boldsymbol{s}$ from the hypothesis of induction. The two paths generated in Steps 2 and 3 consist of nodes on different class paths that share only $\boldsymbol{s}$, sub paths in different sub graphs, and destination node $\boldsymbol{d}_{2n-3}$. Hence, they are disjoint with each other. These paths do not have the nodes in $BR_{2n-3}n$ except for $\boldsymbol{s}$ and $\boldsymbol{d}_{2n-3}$. Hence, they are also disjoint from the paths generated in Step 1.

The time complexity of Step 1 is $T(n-1) + L(n-1) \times O(n)$. The sum of the path lengths generated in Step 1 is $L(n-1)$, and the length of the maximum path in Step 1 is at most $l(n-1)$. The time complexities of Steps 2 and 3 are both $O(n^2)$, and the sums of the lengths of the two paths obtained in Steps 2 and 3 are both $O(n)$. The length of the maximum path generated in Step 2 and 3 is at most $2n - 3$. Hence, the total time complexity of Procedure 2 is $T(n-1)+L(n-1)\times O(n)+O(n^2)$. The sum of the path lengths and the length of the maximum path obtained by Procedure 2 is $L(n-1) + O(n)$ and at most $2n - 3$, respectively. □

Lemma 4: The paths generated by Procedure 3 are disjoint except for the source node. The time complexity of Procedure 3 is $T(n-1) + O(n^4)$. The sum of the path lengths is $L(n-1) + O(n^2)$, and the length of the maximum path is at most $l(n-1) + \max\{\lfloor n/2 \rfloor + 2, n-2\}$.

(Proof) The paths constructed in Steps 2 and 3 consist of different class paths, different class paths followed by edges from their terminal nodes to different destinations, or a class path followed by two edges, $(\boldsymbol{c}, \boldsymbol{u})$ and $(\boldsymbol{u}, \boldsymbol{d}_{2n-3})$. Node $\boldsymbol{u}$, if any, is selected so as not to be shared by other paths. Hence, any pair of these paths are disjoint with each other. In Step 4, if one path is generated, it consists of a class path followed by a path in $BR_{n-1}1$. The path may be still followed by another class path, if necessary. In either case, the path is connected to the previously constructed path at the node that is first encountered. Hence, it is disjoint from other paths except for one that shares $\boldsymbol{s}$. If two paths are generated in Step 4, we can prove that they are disjoint with each other and also disjoint from other paths except for $\boldsymbol{s}$, similar to Step 6. The two paths generated in Step 6 are disjoint except for $\boldsymbol{s}$ because they only share $\boldsymbol{s}$. Nodes $\boldsymbol{v}$ and $\boldsymbol{w}$ are the first encountered nodes that are on other paths in different sub graphs. Hence, they are disjoint from other paths that do not include $\boldsymbol{v}$ or $\boldsymbol{w}$. The paths generated in Step 7 are disjoint from induction hypothesis. These paths are inside $BR_{n-1}n$, and they are connected to the previously constructed paths inside $BR_{n-1}n$. Hence the connected paths are also disjoint.

The time complexity of Step 1 is $O(n^3)$ that is necessary to obtain C. In Step 2, finding the $\boldsymbol{c}_i$'s requires $O(n^4)$. The time complexity of Step 3 is $O(n^3)$. The sum of the lengths of the paths obtained in Step 3 is $O(n^2)$. In Step 4, finding $\boldsymbol{v}$ and $\boldsymbol{w}$ is governing, and it requires $O(n^3)$ time complexity. The sum of the lengths of the one or two paths generated in Step 4 is $O(n)$. The time complexity of Step 5 is $O(n^2)$. In Step 6, the time complexity is $O(n^3)$, and the sum of the path lengths is $O(n)$. Finally, the time complexity in Step 7 is $T(n-1)$, and the sum of the path lengths is $L(n-1)$. On the other hand, the length of the maximum path is estimated as follows. The generated path that does not include the node in $BR_{n-1}n$ other than $\boldsymbol{s}$ has the length of at most $(n-2)+(n-2) + \lfloor n/2 \rfloor + 2$ if the shorter path is selected in Step 3 or at most $(n-2)+(n-2)+(n-3)$ otherwise. The other path that includes the node inside $BR_{n-1}n$ other than $\boldsymbol{s}$ has the length of at most $l(n-1) + \lfloor n/2 \rfloor + 2$ if the shorter path is selected in Step 3 or at most $l(n-1) + (n-2)$ otherwise. Hence, the length of the maximum path generated by Procedure 3 is at most $l(n-1) + \max\{\lfloor n/2 \rfloor + 2, n-2\}$. □

5. Computer Experiment

To evaluate the average performance of our algorithm, we conducted a computer experiment in which we repeated following steps at least 1000 times for random combinations of destination nodes for each n between 3 and 50.

1. In an n-bi-rotator graph, fix source node $\boldsymbol{s}$ to the identity permutation $(1, 2, \cdots, n)$ taking advantage of its symmetric property.
2. Set $2n - 3$ destination nodes other than $\boldsymbol{s}$.
3. Invoke our algorithm, and measure the execution time and the path lengths.

We implemented our algorithm in the functional programming language Haskell and compiled the program using the Glasgow Haskell Compiler (`ghc`) with the `-O` and `-fglasgow-exts` options. We executed the program on a computer with a Celeron 400-MHz CPU, 128 MB of memory, and the FreeBSD 3.5.1 operating system.

Figures 13 and 14 show the results for the execution time and sum of the path lengths, respectively. In each figure, the horizontal axis represents the n of the n-bi-rotator graph. The vertical axis in Fig. 13 represents the average execution time in seconds while that in Fig. 14 represents the average sum of the path lengths obtained by our algorithm.

These figures show that our algorithm obtains $2n - 3$ paths from the source node to the destination nodes that are node-disjoint except for the source node, that it has an average execution time of $O(n^{3.9})$ for an n-bi-rotator graph, and that the average sum of these path lengths is $O(n^{3.0})$.

Figure 15 shows the result for the length of the maximum path. There is a large gap between the estimated and the measured values. This gap may be caused by the too small number of sampling. However, there is a chance that we have omitted some available conditions to improve the estimation.

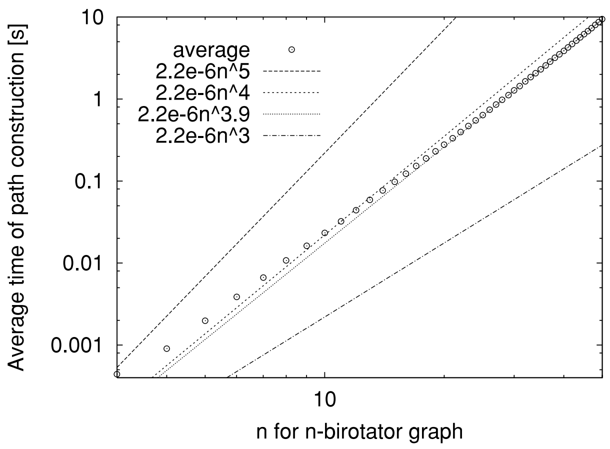

Fig. 13 Execution time.

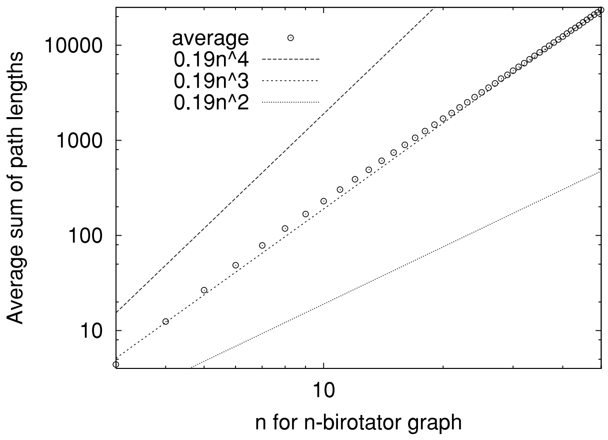

Fig. 14 Average sum of path lengths.

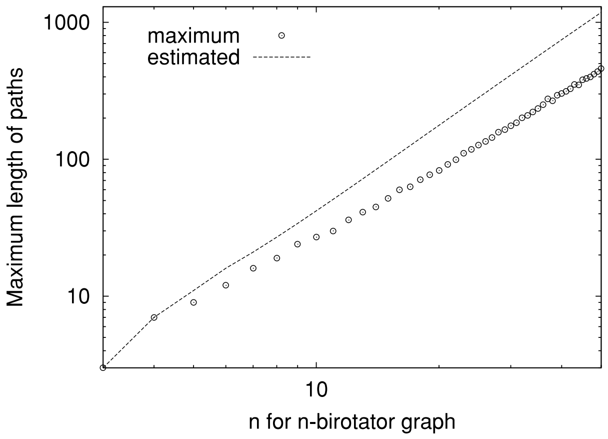

Fig. 15 Length of maximum path.

6. Conclusion

We have described an algorithm for solving the node-to-set disjoint paths problem in an n-bi-rotator graph. Its time complexity, the sum of path lengths, and the length of the maximum path are $O(n^5)$, $O(n^3)$, and $O(n^2)$, respectively. Computer experiment showed that the average execution time of the algorithm was $O(n^{3.9})$ and the average sum of the path lengths was $O(n^{3.0})$. Future work includes improving the algorithm to generate shorter paths more quickly, and improving the estimated value for the maximum path length.

Acknowledgements

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. 16500015.

References

[1] S.B. Akers and B. Krishnamurthy, "A group theoretic model for symmetric interconnection networks," IEEE Trans. Comput., vol.38, no.4, pp.555–566, April 1989.

[2] S.G. Akl and K. Qiu, "Parallel minimum spanning forest algorithms on the star and pancake interconnection networks," Proc. Joint Conference Vector and Parallel Processing, pp.565–570, Sept. 1992.

[3] S.G. Akl and K. Qiu, "A novel routing scheme on the star and pancake interconnection networks and its applications," Parallel Comput., vol.19, no.1, pp.95–101, Jan. 1993.

[4] S.G. Akl, K. Qiu, and I. Stojmenović, "Fundamental algorithms for the star and pancake interconnection networks with applications to computational geometry," Networks, vol.23, no.4, pp.215–226, July 1993.

[5] P. Berthomé, A. Ferreira, and S. Perennes, "Optimal information dissemination in star and pancake networks," IEEE Trans. Parallel Distrib. Syst., vol.7, no.12, pp.1292–1300, Dec. 1996.

[6] P.F. Corbett, "Rotator graphs: An efficient topology for point-to-point multiprocessor networks," IEEE Trans. Parallel Distrib. Syst., vol.3, no.5, pp.622–626, Sept. 1992.

[7] L. Garfgano, U. Vaccaro, and A. Vozella, "Fault tolerant routing in the star and pancake interconnection networks," Inf. Process. Lett., vol.45, no.6, pp.315–320, June 1993.

[8] 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, April 1997.

[9] Y. Hamada, F. Bao, A. Mei, and Y. Igarashi, "Nonadaptive fault-tolerant file transmission in rotator graphs," IEICE Trans. Fundamentals, vol.E79-A, no.4, pp.477–482, April 1996.

[10] K. Kaneko and Y. Suzuki, "An algorithm for node-to-set disjoint paths problem in rotator graphs," IEICE Trans. Inf. & Syst., vol.E84-D, no.9, pp.1155–1163, Sept. 2001.

[11] K. Kaneko and Y. Suzuki, "Node-to-set disjoint paths problem in pancake graphs," IEICE Trans. Inf. & Syst., vol.E86-D, no.9, pp.1628–1633, Sept. 2003.

[12] H.-R. Lin and C.-C. Hsu, "Topological properties of bi-rotator graphs," IEICE Trans. Inf. & Syst., vol.E86-D, no.10, pp.2172–2178, Oct. 2003.

[13] K. Qiu, H. Meijer, and S.G. Akl, "Parallel routing and sorting on the pancake network," Proc. Int'l Conf. Computing and Information, pp.360–371, May 1991.

[14] M.O. Rabin, "Efficient dispersal of information for security, load balancing, and fault tolerance," J. ACM, vol.36, no.2, pp.335–348, Feb. 1989.

[15] Y. Suzuki and K. Kaneko, "An algorithm for node-disjoint paths in pancake graphs," IEICE Trans. Inf. & Syst., vol.E86-D, no.3, pp.610–615, March 2003.

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