IEICE TRANS. INF. & SYST., VOL.E88–D, NO.7 JULY 2005

PAPER

Internally-Disjoint Paths Problem in Bi-Rotator Graphs

Keiichi KANEKO[†a)], ***Member***

SUMMARY A rotator graph was proposed as a topology for interconnection networks of parallel computers, and it is promising because of its small diameter and small degree. However, a rotator graph is a directed graph that sometimes behaves harmfully when it is applied to actual problems. A bi-rotator graph is obtained by making each edge of a rotator graph bi-directional. In a bi-rotator graph, average distance is improved against a rotator graph with the same number of nodes. In this paper, we give an algorithm for the container problem in bi-rotator graphs with its evaluation results. The solution achieves some fault tolerance such as file distribution based information dispersal technique. The algorithm is of polynomial order of n for an n-bi-rotator graph. It is based on recursion and divided into two cases according to the position of the destination node. The time complexity of the algorithm and the maximum length of paths obtained are estimated to be $O(n^3)$ and $4n-5$, respectively. Average performance of the algorithm is also evaluated by computer experiments.

key words: *container problem, internally-disjoint paths, bi-rotator graphs, fault tolerance, parallel computation*

1. Introduction

Recently, research on parallel and distributed computing is becoming more important. Moreover, many studies on so-called massively parallel processing systems are eagerly conducted. Therefore, many complex topologies [1], [7] based on Cayley graphs are proposed for interconnection networks instead of simple networks such as a hypercube, a mesh and so on. There are many research activities concerning them [2]–[6], [8]–[14], [16], [18]–[20]. A rotator graph [7] is one such topology and it is very promising because of its small diameter and degree. However, a rotator graph is sometimes inadequate to solve practical problems because it is directed.

A bi-rotator graph [15] is obtained by making each edge of a rotator graph bi-directional. The average diameter is improved by this modification. One of the unsolved issues concerning this topology is the container problem: for a pair of nodes $\boldsymbol{s}$ and $\boldsymbol{d}$ in a k-connected graph $G = (V, E)$, to find k paths between $\boldsymbol{s}$ and $\boldsymbol{d}$ that are node-disjoint except for $\boldsymbol{s}$ and $\boldsymbol{d}$. The container problem is one of the important issues [6], [8], [11], [14], [18]–[20] in designing parallel and distributed computing systems as well as the node-to-set disjoint paths problem [5], [10], [12], [13], [17]. In this paper, the terms ‘disjoint’ and ‘internally disjoint’ are used to express ‘node-disjoint’ and ‘node-disjoint except for source and destination.’

In general, a container can be obtained by using the algorithm for the maximum-flow problem in polynomial-order time of the number of nodes $|V|$ in the graph. In an n-bi-rotator graph, the number of nodes is equal to $n!$. Hence, the complexity of this approach is impractical. In this paper, we give an algorithm of polynomial order of n instead of $n!$. We estimate the theoretical performance of the algorithm. We also evaluate its average performance by computer experiment.

The rest of this paper is structured as follows. Section 2 gives definitions and auxiliary algorithms. Section 3 explains our algorithm in detail. In Sect. 4, we give a proof of correctness of our algorithm and estimations of its complexities. We conduct computer experiment in Sect. 5. Section 6 describes the conclusion and the future work.

2. Preliminaries

In this section, we first give a definition of a bi-rotator graph and its properties. Next some auxiliary algorithms including a simple unicast routing are presented.

2.1 Definitions

Definition 1: For an arbitrary permutation $\boldsymbol{u} = (u_1, u_2, \cdots, u_n)$ of n symbols of $1, 2, \cdots, n$ and an 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 a same rotation operation. Therefore there are $2n-3$ 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 of $1, 2, \cdots, n$. The node whose address is $\boldsymbol{u} = (u_1, u_2, \cdots, u_n)$ is adjacent to the nodes whose addresses are elements of the set $\{R_i^+(\boldsymbol{u}), R_i^-(\boldsymbol{u}) \mid 2 \le i \le n\}$.

Table 1 shows comparisons of an n-bi-rotator graph BR_n with other topologies. In the table, T_n, Q_n, $B(n,k)$, and $K(n,k)$ represent an $n \times n$-torus, an n-dimensional hypercube, an (n,k)-de Bruijn graph, and an (n,k)-Kautz graph, respectively. Up to the present, the average distance of an n-bi-rotator graph is unknown while its diameter is $n-1$. As

Manuscript received October 5, 2004.
Manuscript revised February 22, 2005.

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

DOI: 10.1093/ietisy/e88–d.7.1678

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

	#Nodes	Degree	Diameter	Integr. Ratio†
BR_n	$n!$	$2n-3$	$n-1$	$\frac{n!}{(n-1)(2n-3)}$
T_n	n^2	4	$2\lfloor n/2 \rfloor$	$n^2/8\lfloor n/2 \rfloor$
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	$\frac{n^{k-1}+n^{k-2}}{k}$

†: #Nodes/(Degree × Diameter)

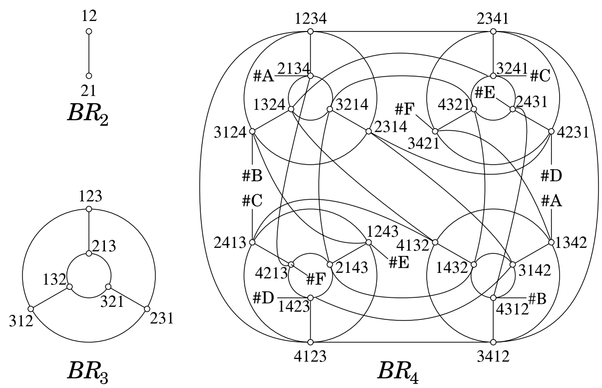


Fig. 1 Examples of 2- to 4- bi-rotator graphs.

for the integration ratio, an n-bi-rotator graph is inferior to an (n,k)-de Bruijn graph and an (n,k)-Kautz graph. However, a bi-rotator graph has a recursive property that is advantageous to execute algorithms based on the divide-and-conquer method in parallel.

Figure 1 shows examples of 2- to 4- bi-rotator graphs. Note that in this figure an address $(u_1, u_2, \cdots, u_n)$ is denoted by $u_1 u_2 \cdots u_n$.

Definition 3: In an n-bi-rotator graph, a sub graph induced by nodes that have a common symbol k at the right-most positions of their addresses forms an $(n-1)$-bi-rotator graph. This sub bi-rotator graph is denoted by $BR_{n-1}k$ by indexing the common symbol k.

2.2 Algorithm A

Here we give an auxiliary algorithm for BR_n in Fig.2 that establishes a path between an arbitrary pair of nodes $\boldsymbol{s}$ and $\boldsymbol{d}$ in polynomial time of n.

We assume that $\boldsymbol{s} = (s_1, s_2, \cdots, s_n)$ and $\boldsymbol{d} = (d_1, d_2, \cdots, d_n)$, and introduce an order relation defined by $d_1 \prec d_2 \prec \cdots \prec d_n$. We also assume that a relation $j \succ i$ holds if and only if $i \prec j$ holds.

It is known that a path generated by Algorithm A from $\boldsymbol{s}$ to $\boldsymbol{d}$ and a path generated by the same algorithm from $\boldsymbol{d}$ to $\boldsymbol{s}$ are internally disjoint [15].

2.3 Algorithm B

For any nodes $\boldsymbol{x}_1, \boldsymbol{x}_2, \boldsymbol{y}_1, \boldsymbol{y}_2$ in BR_n $(n \geq 3)$ where $\boldsymbol{x}_1 \neq \boldsymbol{x}_2$, $\boldsymbol{y}_1 \neq \boldsymbol{y}_2$, we give an algorithm that obtains two disjoint paths each of which has one terminal in $X = \{\boldsymbol{x}_1, \boldsymbol{x}_2\}$ and the other in $Y = \{\boldsymbol{y}_1, \boldsymbol{y}_2\}$ in polynomial time of n.

```
procedure A(s, d)
  (* For d = (d1, d2, ..., dn), the order
     d1 ≺ d2 ≺ ... ≺ dn is introduced. *)
begin
  P := [s];
  Find k such that sk ≻ sk+1 ≺ sk+2 ≺ ... ≺ sn;
  for i := 1 to k do begin
    Find the smallest h such that
      s1 ≺ sh+1 ≺ sh+2 ≺ ... ≺ sn;
    s := R+h(s);
    P := P ++ [s]
  end;
  Output P
end;
```

Fig. 2 Algorithm A.

Step 1 If $X = Y$, then paths $[\boldsymbol{x}_1]$ and $[\boldsymbol{x}_2]$ are already constructed. We output them and terminate. Otherwise, if $X \cap Y \neq \emptyset$, let $\tilde{\boldsymbol{x}} = X \cap Y$, $\boldsymbol{x} = X - \{\tilde{\boldsymbol{x}}\}$ and $\boldsymbol{y} = Y - \{\tilde{\boldsymbol{x}}\}$, obtain two internally disjoint paths between $\boldsymbol{x}$ and $\boldsymbol{y}$ by using Algorithm A to select one of them that does not include $\tilde{\boldsymbol{x}}$, output the path with $[\tilde{\boldsymbol{x}}]$, and terminate.

Step 2 Construct two internally disjoint paths P_1 and P_2 between $\boldsymbol{x}_1$ and $\boldsymbol{y}_1$. If either of $\boldsymbol{x}_2$ or $\boldsymbol{y}_2$ is not on $P_1 \cup P_2$, then go to Step 3. Otherwise if both of $\boldsymbol{x}_2$ and $\boldsymbol{y}_2$ are on $P_1 \cup P_2$, then there are two disjoint paths between $\boldsymbol{x}_1$ and $\boldsymbol{y}_1$, and $\boldsymbol{x}_2$ and $\boldsymbol{y}_2$, or two disjoint sub paths between $\boldsymbol{x}_1$ and $\boldsymbol{y}_2$, and $\boldsymbol{x}_2$ and $\boldsymbol{y}_1$. Hence, we select them to output and terminate.

Step 3 Construct internally disjoint two paths Q_1 and Q_2 between $\boldsymbol{x}_2$ and $\boldsymbol{y}_2$. If both of $\boldsymbol{x}_1$ and $\boldsymbol{y}_1$ are on $Q_1 \cup Q_2$, we can output two disjoint paths and terminate in the similar way in Step 2.

Step 4 If at least one of Q_1 and Q_2 is disjoint to at least one of P_1 and P_2, then we can output two disjoint paths and terminate.

Step 5 Let Q be one of the paths Q_1 and Q_2 that does not include neither $\boldsymbol{x}_1$ nor $\boldsymbol{y}_1$. In addition, let $\boldsymbol{u}$ and $\boldsymbol{v}$ be the nodes on $Q \cap (P_1 \cup P_2)$ that are nearest to $\boldsymbol{x}_2$ and $\boldsymbol{y}_2$, respectively. If $\boldsymbol{u}$ and $\boldsymbol{v}$ are both on either P_1 or P_2, here we assume that it is P_1, then we construct a path Q' that consists of the sub path of Q from $\boldsymbol{x}_2$ to $\boldsymbol{u}$, the sub path of P_1 from $\boldsymbol{u}$ to $\boldsymbol{v}$, and the sub path of Q from $\boldsymbol{v}$ to $\boldsymbol{y}_2$. The paths P_2 and Q' are disjoint each other. On the other hand, if $\boldsymbol{u}$ and $\boldsymbol{v}$ are on different paths of P_1 and P_2, say $\boldsymbol{u}$ is on P_1 and $\boldsymbol{v}$ is on P_2, then we construct two paths P' and Q' so that P' consists of the sub path of P_2 from $\boldsymbol{x}_1$ to $\boldsymbol{v}$ and the sub path of Q from $\boldsymbol{v}$ to $\boldsymbol{y}_2$, and Q' consists of the sub path of Q from $\boldsymbol{x}_2$ to $\boldsymbol{u}$, and the sub path of P_1 from $\boldsymbol{u}$ to $\boldsymbol{y}_1$. The paths P' and Q' are disjoint each other. Therefore, in either case, two disjoint paths are obtained. We output them and terminate.

2.4 Algorithm C

Finally, for $\boldsymbol{x}_1, \boldsymbol{y}_1, \boldsymbol{y}_2 \in BR_{n-1}h$ and $\boldsymbol{x}_2, \boldsymbol{x}_3, \boldsymbol{y}_3 \in BR_{n-1}k$

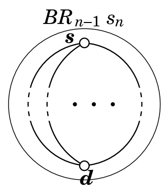


Fig. 3 Recursive application of algorithm.

where $y_1 \neq y_2$, $x_2 \neq x_3$, $h \neq k$ and $n \geq 4$, we give an algorithm that generates three disjoint paths inside $BR_{n-1}h \cup BR_{n-1}k$ each of which has one terminal node in $X = \{x_1, x_2, x_3\}$ and the other in $Y = \{y_1, y_2, y_3\}$ in polynomial time of n.

Step 1 In $BR_{n-1}h$, select a node x_0 that is different from x_1, is not adjacent to y_3, and has an address of $(\cdots, k, h)$ or $(k, \cdots, h)$.

Step 2 For x_0, x_1, y_1, and y_2, apply Algorithm B to obtain two disjoint paths P_0 and P_1. We assume that the first node on P_0 is x_0.

Step 3 Let y_0 be the neighbor node of x_0 in $BR_{n-1}k$. For x_2, x_3, y_3, and y_0, apply Algorithm B to obtain two disjoint paths Q_0 and Q_1. We assume that the last node on Q_0 is y_0.

Step 4 Select an edge (y_0, x_0) to construct a path Q_0 ++P_0. Output three paths P_1, Q_1 and Q_0++P_0 and terminate.

3. Algorithm

3.1 Classification

For a BR_3, the problem is trivial. Hence we assume that $n \geq 4$. Let the source node be $s = (s_1, s_2, \cdots, s_n)$, and the destination node be $d = (d_1, d_2, \cdots, d_n)$. Then we consider the following two cases.

Case 1 The destination node belongs to the same sub bi-rotator graph as the source node ($s_n = d_n$).

Case 2 The destination node is outside of the sub bi-rotator graph to which the source node belongs ($s_n \neq d_n$).

3.2 Case 1

In this section, we give Procedure 1 that obtains $2n-3$ internally disjoint paths between the source node s and the destination node d in case that $s_n = d_n$.

Step 1 In $BR_{n-1}s_n$, apply the algorithm recursively to obtain $2n-5$ internally disjoint paths between s and d. See Fig. 3.

Step 2 Select edges $(s, R_n^+(s))$, $(s, R_n^-(s))$, $(d, R_n^+(d))$, and $(d, R_n^-(d))$.

Step 3 For $R_n^+(s)$, $R_n^-(s)$, $R_n^+(d)$, and $R_n^-(d)$, if either $s_1 = d_1$ or $s_{n-1} = d_{n-1}$ holds, try to establish paths between $R_n^+(s)$ and $R_n^+(d)$, and between $R_n^-(s)$ and $R_n^-(d)$. Otherwise, try to establish paths between $R_n^+(s)$ and $R_n^-(d)$, and between $R_n^-(s)$ and $R_n^+(d)$. If the pair of nodes belong to a same sub graph, apply the auxiliary algorithm A inside the sub graph to obtain a path. Otherwise, select a path between the pair of nodes so that the path does not include any node outside of the sub graphs to which those nodes belong. See Fig. 4.

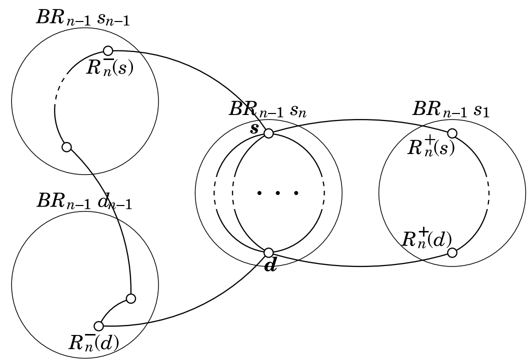


Fig. 4 Construction of two outside paths ($s_1 = d_1, s_{n-1} \neq d_{n-1}$).

3.3 Case 2

In this section, we give Procedure 2 that obtains $2n-3$ internally disjoint paths between the source node s and the destination node d in case that $s_n \neq d_n$.

Step 1 First, name each of neighbor nodes of s as follows:

$$a_i = R_i^+(s) \quad (2 \leq i \leq n), \qquad b_i = R_i^-(s) \quad (3 \leq i \leq n).$$

Next, construct paths from s to sub graphs $BR_{n-1}s_1$, $BR_{n-1}s_2, \cdots, BR_{n-1}s_{n-1}$ that are disjoint except for s as follows. Let u be the other terminal node of the path from s to $BR_{n-1}d_n$.

(In case that $d_n = s_1$)

$a_2 \to R_n^+(a_2)(\in BR_{n-1}s_2)$
$a_3 \to R_2^-(a_3) \to R_n^+(R_2^-(a_3))(\in BR_{n-1}s_3)$
$\vdots$
$a_{n-2} \to R_{n-3}^-(a_{n-2}) \to R_n^+(R_{n-3}^-(a_{n-2}))(\in BR_{n-1}s_{n-2})$
$a_{n-1} \to R_n^+(a_{n-1})(\in BR_{n-1}s_2)$
$a_n(\in BR_{n-1}s_1)$
$b_3 \to R_n^+(b_3)(\in BR_{n-1}s_3)$
$b_4 \to R_n^+(b_4)(\in BR_{n-1}s_4)$
$\vdots$
$b_{n-1} \to R_n^+(b_{n-1})(\in BR_{n-1}s_{n-1})$
$b_n(\in BR_{n-1}s_{n-1})$
See Fig. 5.

(In case that $d_n = s_{n-1}$)

$a_2 \to R_n^+(a_2)(\in BR_{n-1}s_2)$
$a_3 \to R_2^-(a_3) \to R_n^+(R_2^-(a_3))(\in BR_{n-1}s_3)$
$\vdots$
$a_{n-2} \to R_{n-3}^-(a_{n-2}) \to R_n^+(R_{n-3}^-(a_{n-2}))(\in BR_{n-1}s_{n-2})$

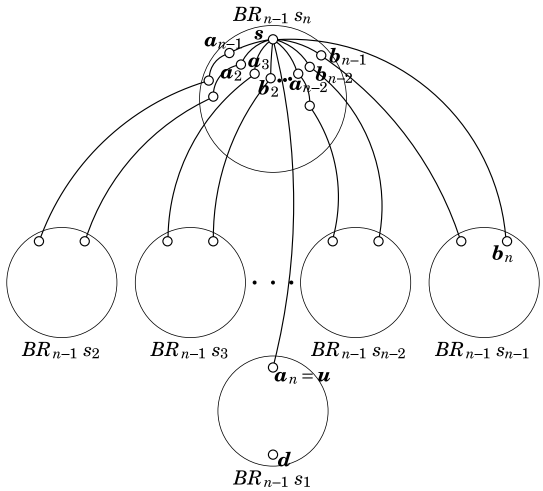


Fig. 5 Construction of disjoint paths from the source node to sub graphs ($d_n = s_1$).

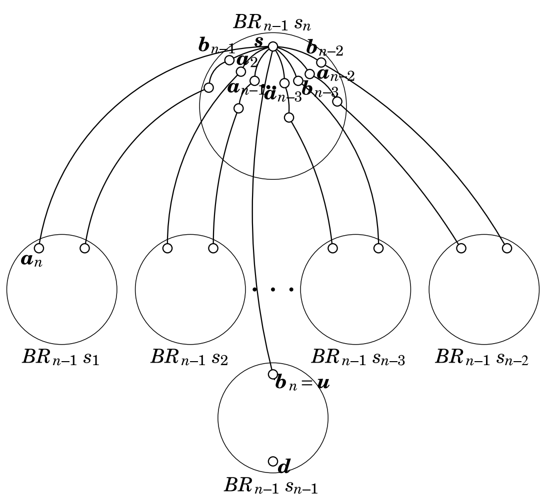


Fig. 6 Construction of disjoint paths from the source node to sub graphs ($d_n = s_{n-1}$).

$a_{n-1} \to R^+_n(a_{n-1}) (\in BR_{n-1}s_2)$
$a_n (\in BR_{n-1}s_1)$
$b_3 \to R^+_n(b_3) (\in BR_{n-1}s_3)$
$b_4 \to R^+_n(b_4) (\in BR_{n-1}s_4)$
$\vdots$
$b_{n-2} \to R^+_n(b_{n-2}) (\in BR_{n-1}s_{n-2})$
$b_{n-1} \to R^-_2(b_{n-1}) \to R^+_n(R^-_2(b_{n-1})) (\in BR_{n-1}s_1)$
$b_n (\in BR_{n-1}s_{n-1})$
See Fig. 6.

(In case that $d_n = s_j$ ($j \neq 1, n-1$))
$a_2 \to R^+_n(a_2) (\in BR_{n-1}s_2)$
$a_3 \to R^-_2(a_3) \to R^+_n(R^-_2(a_3)) (\in BR_{n-1}s_3)$
$\vdots$
$a_{j-1} \to R^-_{j-2}(a_{j-1}) \to R^+_n(R^-_{j-2}(a_{j-1})) (\in BR_{n-1}s_{j-1})$
$a_j \to R^-_{j+1}(a_j) \to R^+_n(R^-_{j+1}(a_j)) (\in BR_{n-1}s_{j+1})$

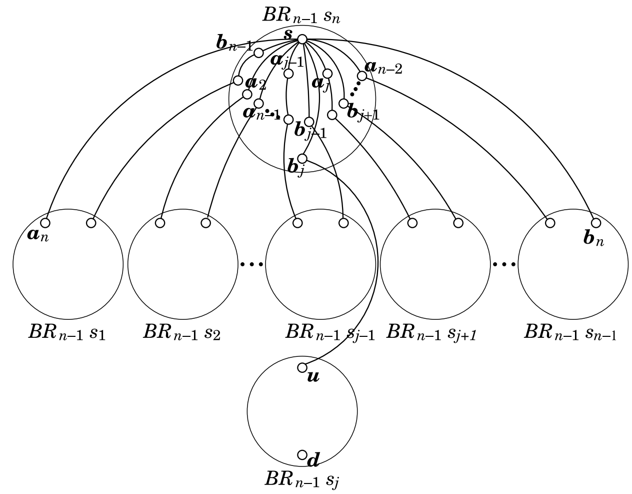


Fig. 7 Construction of disjoint paths from the source node to sub graphs ($d_n = s_j$, $j \neq 1, n-1$).

$\vdots$
$a_{n-2} \to R^-_{n-1}(a_{n-2}) \to R^+_n(R^-_{n-1}(a_{n-2})) (\in BR_{n-1}s_{n-1})$
$a_{n-1} \to R^+_n(a_{n-1}) (\in BR_{n-1}s_2)$
$a_n (\in BR_{n-1}s_1)$
$b_3 \to R^+_n(b_3) (\in BR_{n-1}s_3)$
$b_4 \to R^+_n(b_4) (\in BR_{n-1}s_4)$
$\vdots$
$b_{n-2} \to R^+_n(b_{n-2}) (\in BR_{n-1}s_{n-2})$
$b_{n-1} \to R^-_2(b_{n-1}) \to R^+_n(R^-_2(b_{n-1})) (\in BR_{n-1}s_1)$
$b_n (\in BR_{n-1}s_{n-1})$
See Fig. 7.

Step 2 As similar to Step 1, construct paths from the destination node d to sub graphs $BR_{n-1}d_1, BR_{n-1}d_2, \cdots, BR_{n-1}d_{n-1}$ that are disjoint except for d. If $u = d$, then refrain from constructing a path from d to $BR_{n-1}s_n$. Otherwise, that is, if $u \neq d$, then let v be the other terminal node of the path from d to $BR_{n-1}s_n$. See Fig. 8.

Step 3 If a path between s and d is not established yet, construct paths from u to d, and from v to d, and let $\tilde{u}$ and $\tilde{v}$ be the first nodes on the previously constructed paths that are encountered by these paths. Let $\hat{u}$ and $\hat{v}$ be the terminal nodes of these paths other than s and d, and discard sub paths from $\tilde{u}$ to $\hat{u}$ and from $\tilde{v}$ to $\hat{v}$. See Fig. 9.

Step 4 In each sub graph other than $BR_{n-1}s_n$ and $BR_{n-1}d_n$, if even number of paths constructed in Steps 1 and 2 have reached to the sub graph, apply Algorithm A or B to connect terminal nodes appropriately. If there are two sub graphs both of which have three terminal nodes of paths, for these sub graphs, apply Algorithm C to connect these terminal nodes appropriately. See Fig. 10.

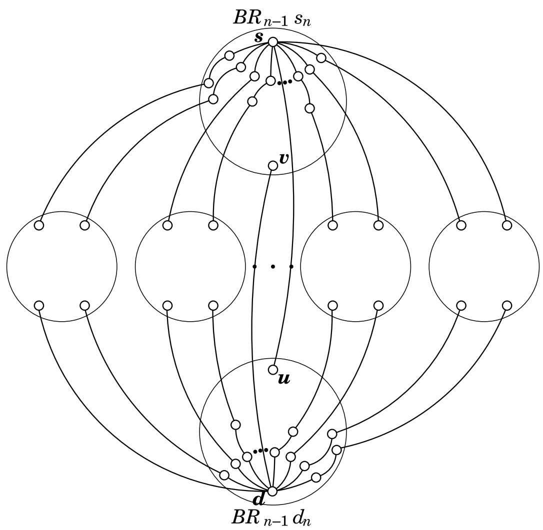


Fig. 8 Construction of disjoint paths from the destination node to sub graphs.

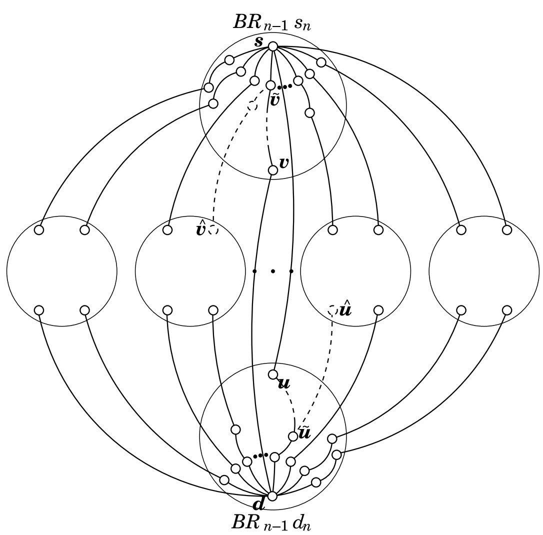


Fig. 9 Construction of paths between $\boldsymbol{s}$ and $\boldsymbol{d}$.

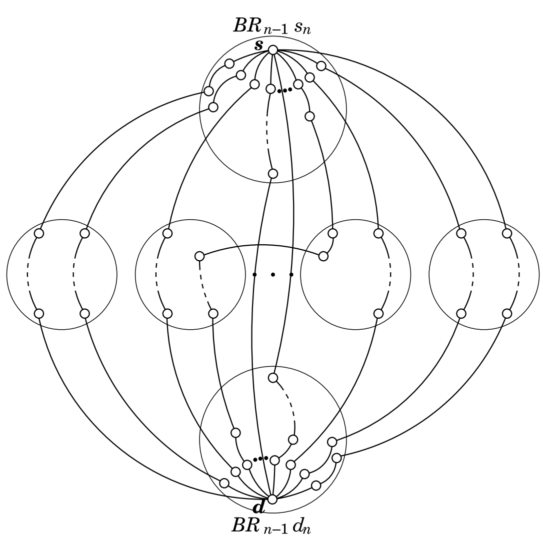


Fig. 10 Construction of paths in sub bi-rotator graphs.

4. Proof of Correctness and Estimations of Complexities

In this section, we give a proof of correctness of our algorithm and estimations of complexities concerning about its execution time and the maximum length of paths generated by our algorithm. Note that we use a linear array to represent an address of a node and all the nodes on the constructed paths are stored in memory.

First we give some lemmas which hold for auxiliary algorithms without proofs.

Lemma 1: For an n-bi-rotator graph, the time complexity of Algorithm A is $O(n^2)$ and the maximum length of paths obtained is $n-1$.

Note that all the internal nodes except for two terminal nodes on the path generated by Algorithm A have different first symbols in their addresses. Hence, if we compare the addresses of nodes from the first symbols, we can decide if any node is on such a path or not in $O(n)$ because only three nodes on the path match with the first symbol of the node at most. Therefore, the time complexity of Algorithm B is $O(n^2)$.

Lemma 2: For an n-bi-rotator graph, Algorithm B generates disjoint two paths and its time complexity is $O(n^2)$. The maximum length of paths obtained by Algorithm B is $2n-4$.

Lemma 3: For an n-bi-rotator graph, Algorithm C generates disjoint two paths and its time complexity is $O(n^2)$. The maximum length of paths obtained by Algorithm C is $4n-11$.

Next we show a theorem for our algorithm as well as lemmas which are necessary to prove the theorem in this order.

Theorem 1: The paths generated by our algorithm are internally disjoint. Let $T(n)$ represent the time complexity of the algorithm for an n-bi-rotator graph. Then $T(n) = O(n^3)$. Let $L(n)$ represent the maximum length of the paths. Then $L(n) = 4n-5$.
(Proof) Based on the facts that $T(3) = O(1)$ and $L(3) = 3$ hold, and induction on n, this theorem can be proved from the following two lemmas. □

Lemma 4: The paths generated by Procedure 1 are internally disjoint. The time complexity of Procedure 1 is $T(n-1) + O(n^2)$, and the maximum length of the paths is $\max\{L(n-1), n+2\}$.
(Proof) The paths obtained in Step 1 are internally disjoint from induction hypothesis. Two paths between $\boldsymbol{s}$ and $\boldsymbol{d}$ generated in Steps 2 and 3 consist of nodes in different sub graphs other than $BR_{n-1}s_n$ except for $\boldsymbol{s}$ and $\boldsymbol{d}$. Therefore, all paths obtained by Procedure 1 are internally disjoint.

The time complexity of Step 1 is $T(n-1)$ and the maximum length of paths generated in Step 1 is $L(n-1)$. The time complexities of Step 2 and Step 3 are $O(n)$ and $O(n^2)$, respectively. The maximum length of the paths generated in Steps 2 and 3 is $n+2$. Hence the time complexity of Procedure 1 is $T(n-1)+O(n^2)$, and the maximum length of paths generated by Procedure 1 is $\max\{L(n-1), n+2\}$. □

Lemma 5: The paths generated by Procedure 2 are internally disjoint. The time complexity of Procedure 2 is $O(n^3)$, and the maximum length of the paths is $4n-5$.
(Proof) The final edges of the paths obtained in Step 1 are all generated by the positive rotation operation R_n^+. Then if we show that the nodes on the paths in $BR_{n-1}s_n$ are all different each other, it is proved that the paths obtained in Step 1 are disjoint except for the source node. First, the nodes $\boldsymbol{a}_i$'s and $\boldsymbol{b}_j$'s are all different neighbor nodes of $\boldsymbol{s}$. Hence, we show that other nodes on the paths in $BR_{n-1}s_n$ are different each other and they are not adjacent to $\boldsymbol{s}$. In case that $d_n = s_1$, other nodes on the paths are obtained by applying R_{i-1}^- to $\boldsymbol{a}_i$. Hence, these nodes have addresses which can be obtained by exchanging s_1 and s_i in the address of the source node $\boldsymbol{s}$ and they are different each other and they are not adjacent to $\boldsymbol{s}$. In case that $d_n = s_{n-1}$, proof is similar to the case of $d_n = s_1$ except for $R_2^-(\boldsymbol{b}_{n-1})$. As for $R_2^-(\boldsymbol{b}_{n-1})$, if we focus on the fact that it has an address which is obtained by inserting s_{n-1} between s_1 and s_2 in the address of $\boldsymbol{s}$, it is deduced that the node is not adjacent to $\boldsymbol{s}$ and it is different from other nodes on the paths. In case that $d_n = s_j$, $j \neq 1, n-1$, if we focus on the fact that for $\boldsymbol{a}_i$ ($3 \leq i \leq n-1$, $i \neq j$), paths via nodes that have addresses obtained by exchanging s_1 and s_i or s_{i+1} in the address of $\boldsymbol{s}$ are constructed, proof can be obtained in a similar way in the case of $d_n = s_{n-1}$. In Step 2, we can prove that the paths generated are disjoint except for the destination node in a similar way in Step 1. In Step 3, the paths established between $\boldsymbol{s}$ and $\boldsymbol{d}$ by discarding sub paths and other paths are internally disjoint. Finally, paths generated in Step 4 are disjoint from lemmas 2 and 3, and the paths obtained by connecting these paths and the paths generated in Steps 1 and 2 are also internally disjoint.

The time complexities of Steps 1 and 2 are both $O(n^2)$, and the maximum lengths of paths are both 3. The time complexity of Step 3 and the maximum length of paths generated in Step 3 are $O(n^2)$ and $n-2$, respectively. From lemmas 2 and 3, the time complexity of Step 4 is $O(n^3)$ and the maximum length of paths generated in Step 4 is $4n-11$. Therefore the whole time complexity is $O(n^3)$ and the maximum length of paths generated in Procedure 2 is $4n-5$. □

5. Computer Experiment

To evaluate the performance of our algorithm, for each n between 3 and 50 we selected 10,000 random combinations of the source and destination nodes to apply our algorithm and measured the average execution time and the maximum path lengths.

The algorithm is implemented by a functional programming language Haskell. The program is compiled by `ghc` (glasgow Haskell compiler) with `-O` and `-fglasgow-exts` options. The experiment is conducted on a machine whose OS is Red Hat Linux 7.2, CPU is Pentium III 700 MHz, and memory unit is 256 MB.

Figures 11 and 12 show the results of the average execution time and the maximum path length, respectively. In each figure, the horizontal axis represents the value of n. The vertical axes of Figs. 11 and 12 represent the average execution time in second and the maximum length of paths obtained by our algorithm, respectively. In case of $n = 50$, the average execution time is 1.747×10^{-1} and the standard deviation is 2.873×10^{-2}. Hence the 95-percent confidence interval is $[1.742 \times 10^{-1}, 1.753 \times 10^{-1}]$, which is small enough.

From these figures, we can conclude that for an n-bi-rotator graph, our algorithm generates $2n-3$ internally disjoint paths in the average execution time $O(n^{3.0})$ and the maximum length of these paths is $2n+2$ in practice.

In general, the maximum length of paths generated by Algorithm C is $4n-11$. However, in the implementation of Step 1 of Algorithm C, we try to find the node $\boldsymbol{x}_0$ so that its neighbor node $\boldsymbol{y}_0$ becomes either $\boldsymbol{x}_1$ or $\boldsymbol{x}_2$, or becomes a neighbor node of $\boldsymbol{x}_1$ or $\boldsymbol{x}_2$. Though it is not proved that such node can be always selected and the adjacent nodes are always connected in Step 3 of Algorithm C, it worked well

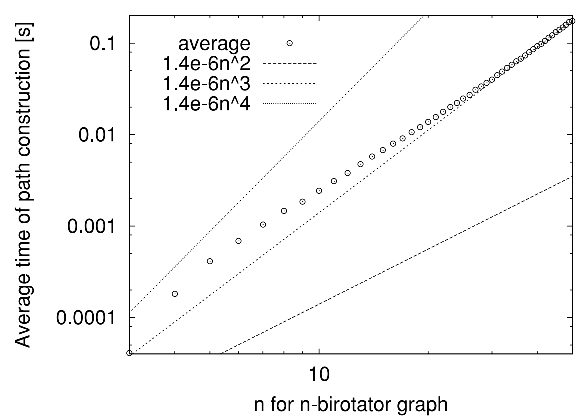


Fig. 11 Average execution time of our algorithm.

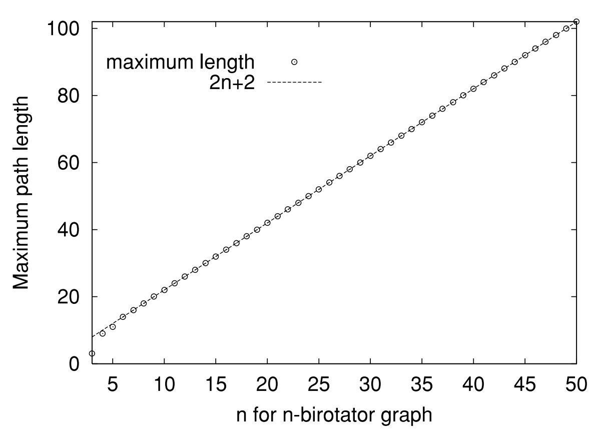


Fig. 12 Maximum length of paths obtained by our algorithm.

in critical cases in the experiment. Therefore the maximum length of the paths generated by Algorithm C is $2(n - 1) - 4 + 2 = 2n - 4$. Hence the maximum length of the paths is $2n + 2$ as a whole.

6. Conclusion and Future Work

In this paper, we have proposed an algorithm that solves the internally-disjoint paths problem in an n-bi-rotator graph. The algorithm is divided into two cases and it uses three auxiliary algorithms. Theoretical values for its time complexity and the maximum length of paths obtained by the algorithm are estimated to be $O(n^3)$ and $4n - 5$, respectively. Computer experiment showed that the disjoint paths are obtained in $O(n^{3.0})$ time and the maximum length of them is $2n + 2$. The future work includes improvement of the algorithm so that the shorter paths are obtained in shorter time. The strict estimation of the maximum length of the disjoint paths is also included in the future work.

Acknowledgement

This study is partly supported by Grant-in-Aid for Scientific Research (C) of Japan Society for the Promotion of Science under the 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] B. Bose, B. Broeg, Y. Kwon, and Y. Ashir, "Lee distance and topological properties of k-ary n-cubes," IEEE Trans. Comput., vol.44, no.8, pp.1021–1030, Aug. 1995.

[7] 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.

[8] M. Dietzfelbinger, S. Madhavapeddy, and I.H. Sudborough, "Three disjoint path paradigms in star networks," Proc. 3rd IEEE Symp. Parallel and Distributed Processing, pp.400–406, 1991.

[9] 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.

[10] 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.

[11] 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.

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

[13] K. Kaneko 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.

[14] K. Kaneko and Y. Suzuki, "Node-to-node internally disjoint paths problem in bubble-sort graphs," Proc. 10th Pacific Rim Int'l Symp. Dependable Computing, pp.173–182, March 2004.

[15] 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.

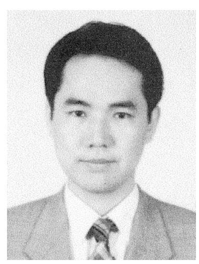
[16] K. Qiu, H. Meijer, and S.G. Akl, "Parallel routing and sorting on the pancake network," Proc. Int'l Conf. Computing and Information, pp.360–371, May 1991.

[17] 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.

[18] Y. Saad and M.H. Schultz, "Topological properties of hypercubes," IEEE Trans. Comput., vol.37, no.7, pp.867–872, July 1988.

[19] 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.

[20] Y. Suzuki, K. Kaneko, and M. Nakamori, "Container problem in substring reversal graphs," Proc. Int'l Symp. Parallel Architectures, Algorithms and Networks, pp.563–568, May 2004.



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 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 also a member of ACM, IPSJ, and JSSST.