IEICE TRANS. INF. & SYST., VOL.E86–D, NO.12 DECEMBER 2003

PAPER *Special Issue on Dependable Computing*

An Algorithm for Node-to-Set Disjoint Paths Problem in Burnt Pancake Graphs

Keiichi KANEKO[†], ***Regular Member***

SUMMARY A burnt pancake graph is a variant of Cayley graphs and its topology is suitable for massively parallel systems. However, for a burnt pancake graph, there is much room for further research. Hence, in this study, we focus on n-burnt pancake graphs and propose an algorithm to obtain n disjoint paths from a source node to n destination nodes in polynomial order time of n, n being the degree of the graph. In addition, we estimate the time complexity of the algorithm and the sum of path lengths. We also give a proof of correctness of the algorithm. Moreover, we report the results of computer simulation to evaluate the average performance of the algorithm.
key words: *burnt pancake graph, disjoint paths, polynomial algorithm, fault tolerance, routing algorithm*

1. Introduction

As studies of parallel and distributed processing proceed, massively parallel systems which connect many processing elements have been proposed. However, a large number of processing elements increases the possibility of the existence of faulty elements in the system, and it is very important to be able to communicate while avoiding these faulty elements. For this reason, one of the essential problems in the area of parallel and distributed processing is to find disjoint paths from one node to a node set [9], [13], [14] as well as to find disjoint paths between two nodes [6], [10], [15].

As an application example of the disjoint paths from a node to a node set, there is the information dispersal algorithm by Rabin [16]. The algorithm divides a file into n pieces and distributes them to n different nodes in the network. Even if m of n pieces are lost, the whole file can be recovered from remaining $n-m$ pieces. When distributing the n pieces to different nodes, it may be possible to avoid unrecoverable damage or loss of data pieces by nodes or link faults on distribution by using the disjoint paths.

A burnt pancake graph [8], [11] is a variant of Cayley graphs [2]–[4], [7], [17] and its topology is suitable for massively parallel systems because its degree and diameter are smaller than a hypercube [17] that has a similar number of nodes. However, for a burnt pancake graph, neither the precise diameter nor an algorithm to obtain shortest paths in polynomial time of the degree is known, like pancake graphs [2]. So there is much room for further research. Hence, in this study, we focus on n-burnt pancake graphs and propose an algorithm to obtain n disjoint paths from a source node to n destination nodes in polynomial order time of n, n being the degree of the graph. In the algorithm, the notion of the traversal class is introduced and all nodes are classified into those classes. Next, the algorithm takes advantage of the recursive structure of a burnt pancake graph, and reduces the generation of paths to $n-1$ destination nodes to the subproblem in the burnt pancake subgraph to which the source node belongs. The path to the remaining one destination node which is disjoint to other paths is obtained by using a path whose nodes belong to a specific traversal class. In addition, we prove the correctness of our algorithm as well as estimate its time complexity and the complexity of the sum of path lengths. Moreover, we conducted computer simulations to evaluate the average performance of our algorithm and report the results of this evaluation.

The rest of this paper is constructed as follows. Chapter 2 introduces requisite definitions. Next, in Chapter 3, we show our algorithm. Then, in Chapter 4, we show the proof of its correctness and estimate its complexity. Chapter 5 describes the computer simulation and shows the average complexities. Finally, we conclude in Chapter 6.

2. Definitions

First we define a signed permutation and a prefix reversal operation.

Definition 1: Consider a permutation $(a_1, a_2, \cdots, a_n)$ of n integers, $1, 2, \cdots, n$, and n signs $b_i \in \{-1, +1\}$ $(1 \leqq i \leqq n)$. Then we call a sequence $\boldsymbol{u} = (a_1 \times b_1, a_2 \times b_2, \cdots, a_n \times b_n)$ a signed sequence of n integers, $1, 2, \cdots, n$.

Definition 2: For a signed permutation $\boldsymbol{u} = (u_1, u_2, \cdots, u_n)$ of n integers, $1, 2, \cdots, n$, we define the prefix reversal operation $P_i(\boldsymbol{u})$ as follows:

$$P_i(\boldsymbol{u}) = (-u_i, -u_{i-1}, \cdots, -u_1, u_{i+1}, \cdots, u_n)$$

In the rest of this paper, we put the negative sign at the top of symbols, $\overline{u}_i$, to save space.

The definition of an n-burnt pancake graph follows.

Manuscript received March 29, 2003.
Manuscript revised July 7, 2003.
[†]The author is with the Faculty of Technology, Tokyo University of Agriculture and Technology, Koganei-shi, 184–8588 Japan.

Definition 3: A burnt pancake graph with degree n, or an n-burnt pancake graph, B_n, has $n! \times 2^n$ nodes, each of which has a unique label which is a signed permutation of $1, 2, \cdots, n$. The node which has a label $\boldsymbol{u} = (u_1, u_2, \cdots, u_n)$ is adjacent to nodes whose labels are elements of the set $\{P_i(\boldsymbol{u}) \mid 1 \leqq i \leqq n\}$.

Table 1 shows comparisons of a burnt pancake graph against other topologies. In this table, P_n, R_n, S_n, $T_{k,n}$, $M_{k,n}$, Q_n, $dB_{n,k}$, and $K_{n,k}$ represent an n-pancake graph, an n-rotator graph [5], an n-star graph [2], a k-ary n-dimensional torus, a k-ary n-dimensional mesh, an n-dimensional hypercube, an (n, k)-de Bruijn graph, and an (n, k)-Kautz graph, respectively. If we define an index

$$\text{(Number of Nodes)}/\{\text{(Degree)} \times \text{(Diameter)}\},$$

B_n is superior to P_n, R_n, S_n, $T_{k,n}$, $M_{k,n}$, and Q_n. Though $dB_{n,k}$ and $K_{n,k}$ are superior to B_n in this index, they do not have symmetry nor recursive structure that are suitable for executing some parallel and distributed programs.

Figure 1 shows an example of a burnt pancake graph. In an n-burnt pancake graph, a subgraph induced by fixing the final integer of labels to k is an $(n-1)$-burnt pancake graph. An n-burnt pancake graph consists of $2n$ disjoint $(n-1)$-burnt pancake graphs. We use $B_{n-1}k$ to denote the burnt pancake subgraph by fixing the final integer to k.

Definition 4: For any node $\boldsymbol{u}$ in an n-burnt pancake graph, a node set produced by alternately applying the operations P_n and P_{n-1} alternatively is called a *traversal class.*

We use $C(\boldsymbol{u})$ to denote a traversal class to which $\boldsymbol{u}$ belongs. A path whose nodes belong to the same traversal class is called a traversal class path.

Table 1 Comparisons of a burnt pancake graph against other topologies.

	#Nodes	Degree	Diameter
B_n	$n! \times 2^n$	n	$\leqq 2n+3$
P_n	$n!$	$n-1$	$\leqq \left\lceil \frac{5(n+1)}{3} \right\rceil$
R_n	$n!$	$n-1$	$n-1$
S_n	$n!$	$n-1$	$\left\lfloor \frac{3(n-1)}{2} \right\rfloor$
$T_{k,n}$	k^n	$2n$	$n\left\lfloor \frac{k}{2} \right\rfloor$
$M_{k,n}$	k^n	$2n$	$n(k-1)$
Q_n	2^n	n	n
$dB_{n,k}$	n^k	n	k
$K_{n,k}$	$n^k + n^{k-1}$	n	k

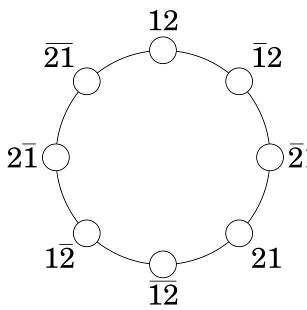

Fig. 1 An example of a 2-burnt pancake graph.

Theorem 1: From the above definitions, for traversal classes of an n-burnt pancake graph where $n \geqq 3$, we derive the following properties.

1. A traversal class C has $4n$ nodes which are connected in a ring structure.
2. There are $(n-1)! \times 2^{n-2}$ traversal classes.
3. Each burnt pancake subgraph overlaps with each traversal class by exactly two adjacent nodes.
4. Among the n neighbor nodes of each node $\boldsymbol{u}$, exactly two nodes belong to $C(\boldsymbol{u})$. The other neighbor nodes belong to different traversal classes from $C(\boldsymbol{u})$ and from each other.

(Proof) Each property is proved as follows:

1. For any node $\boldsymbol{u} = (u_1, u_2, \cdots, u_n)$, alternative applications of P_n and P_{n-1} generate a node sequence $(\overline{u}_n, \overline{u}_{n-1}, \cdots, \overline{u}_1)$, $(u_2, u_3, \cdots, u_n, \overline{u}_1)$, $(u_1, \overline{u}_n, \cdots, \overline{u}_2)$, $(u_3, \cdots, u_n, \overline{u}_1, \overline{u}_2)$, $\cdots$, $(u_{n-1}, u_{n-2}, \cdots, u_1, \overline{u}_n)$, $(\overline{u}_1, \overline{u}_2, \cdots, \overline{u}_n)$, $(u_n, u_{n-1}, \cdots, u_1)$, $(\overline{u}_2, \cdots, \overline{u}_n, u_1)$, $(\overline{u}_1, u_n, \cdots, u_2)$, $(\overline{u}_3, \cdots, \overline{u}_n, u_1, u_2)$, $(\overline{u}_2, \overline{u}_1, u_n, \cdots, u_3)$, $\cdots$, $(\overline{u}_{n-1}, \cdots, \overline{u}_1, u_n)$, $(u_1, u_2, \cdots, u_n)$. This sequence constructs a cycle whose length is $4n$. If we take pairs of elements from the beginning of this sequence two by two, each pair of nodes has a unique integer at their last positions. In addition, the two nodes in a pair are also different. Hence, each node pair belongs to a unique burnt pancake subgraph and the node set construction method results in a simple cycle, that is, a ring structure.
2. The number of nodes is $n! \times 2^n$ and each traversal class has $4n$ nodes. Therefore, the number of traversal classes is $(n-1)! \times 2^{n-2}$.
3. Trivial from the proof of Property 1.
4. From the definition of traversal classes, it follows that neighbor nodes $P_n(\boldsymbol{u})$ and $P_{n-1}(\boldsymbol{u})$ of node $\boldsymbol{u}$ belong to $C(\boldsymbol{u})$. In addition, from Property 3, it follows that there are two nodes in each burnt pancake subgraph which belong to $C(\boldsymbol{u})$. Moreover, all nodes in $N = \{P_1(\boldsymbol{u}), P_2(\boldsymbol{u}), \cdots, P_{n-2}(\boldsymbol{u})\}$ belong to the burnt pancake subgraph that includes $\boldsymbol{u}$. Therefore, the nodes in N do not overlap with those in $C(\boldsymbol{u})$. Additionally, if $C(P_i(\boldsymbol{u})) = C(P_j(\boldsymbol{u}))$ $(1 \leqq i < j \leqq n-2)$ then, from the proof of Property 1, $P_{n-1}(P_i(\boldsymbol{u})) = P_j(\boldsymbol{u})$ holds and this implies that $j = n-1$, which is a contradiction. Hence, all nodes in N belong to different traversal classes.

For instance, a 3-burnt pancake graph, B_3, has the following four traversal classes, C_1, C_2, C_3, and C_4, and each traversal class has twelve nodes.

$$\begin{aligned}
C_1 = \{&(1,2,3),(\overline{3},\overline{2},\overline{1}),(2,3,\overline{1}),(1,\overline{3},\overline{2}),\\
&(3,\overline{1},\overline{2}),(2,1,\overline{3}),(\overline{1},\overline{2},\overline{3}),(3,2,1),\\
&(\overline{2},\overline{3},1),(\overline{1},3,2),(\overline{3},1,2),(\overline{2},\overline{1},3)\}\\
C_2 = \{&(\overline{2},1,3),(\overline{3},\overline{1},2),(1,3,2),(\overline{2},\overline{3},\overline{1}),\\
&(3,2,\overline{1}),(1,\overline{2},\overline{3}),(2,\overline{1},\overline{3}),(3,1,\overline{2}),\\
&(\overline{1},\overline{3},\overline{2}),(2,3,1),(\overline{3},\overline{2},1),(\overline{1},2,3)\}\\
C_3 = \{&(\overline{1},\overline{2},3),(\overline{3},2,1),(\overline{2},3,1),(\overline{1},\overline{3},2),\\
&(3,1,2),(\overline{2},\overline{1},\overline{3}),(1,2,\overline{3}),(3,\overline{2},\overline{1}),\\
&(2,\overline{3},\overline{1}),(1,3,\overline{2}),(\overline{3},\overline{1},\overline{2}),(2,1,3)\}\\
C_4 = \{&(2,\overline{1},3),(\overline{3},1,\overline{2}),(\overline{1},3,\overline{2}),(2,\overline{3},1),\\
&(3,\overline{2},1),(\overline{1},2,\overline{3}),(\overline{2},1,\overline{3}),(3,\overline{1},2),\\
&(1,\overline{3},2),(\overline{2},3,\overline{1}),(\overline{3},2,\overline{1}),(1,\overline{2},3)\}
\end{aligned}$$

As mentioned in the previous section, for an n-burnt pancake graph, there is not yet any known algorithm to find the shortest path between two nodes in polynomial time of n. However, we have an algorithm which finds a path of length at most $3n$ in polynomial time of $O(n^2)$ for any two nodes.

Definition 5: We call the routing algorithm for two nodes $\boldsymbol{s} = (s_1, s_2, \cdots, s_n)$ and $\boldsymbol{d} = (d_1, d_2, \cdots, d_n)$ in an n-burnt pancake graph shown in Fig. 2 a polynomial-time routing algorithm.

Finally, we introduce the definition of the node-to-set disjoint paths problem. The problem can be stated for a k-connected graph that is a graph for which deletions of any set of $k - 1$ nodes cannot make the graph disconnected.

Definition 6: For any node $\boldsymbol{s}$ and a set of nodes $\{\boldsymbol{d}_1, \boldsymbol{d}_2, \cdots, \boldsymbol{d}_k\}$ in a k-connected graph, the node-to-set paths problem is to find k paths from $\boldsymbol{s}$ to $\boldsymbol{d}_i$'s which are node-disjoint except for $\boldsymbol{s}$.

Menger's theorem ensures that for any source node and k arbitrary destination nodes in an arbitrary k-connected graph there exist k paths from the source to the destinations that are disjoint except for the source. The maximal flow algorithm can obtain these k disjoint paths in polynomial time of the number of nodes in the graph. However, this approach is impractical in cases for which the number of nodes is very large.

```
procedure routing($\boldsymbol{s}$,$\boldsymbol{d}$)
begin
  $P$ := [$\boldsymbol{s}$]; /* empty path */
  $\boldsymbol{c}$ := $\boldsymbol{s}$; /* current node */

  for $k$ := $n$ to 1 step $-1$ do begin
    while $c_k$ = $d_k$ do $k$ := $k$ - 1;
    if $k$ = 0 then break;
    find $h$ where $|c_h|$ = $|d_k|$;
    if 1 < $h$ then /* bring $\overline{c}_h$ at the front position */
      begin $\boldsymbol{c}$ := $P_h(\boldsymbol{c})$; $P$ := $P$ ++ [$\boldsymbol{c}$] end;
    if $c_1$ = $d_k$ then /* make $\overline{c}_1$ = $d_k$ */
      begin $\boldsymbol{c}$ := $P_1(\boldsymbol{c})$; $P$ := $P$ ++ [$\boldsymbol{c}$] end;
    $\boldsymbol{c}$ := $P_k(\boldsymbol{c})$; $P$ := $P$ ++ [$\boldsymbol{c}$]
  end
end;
```

Fig. 2 A polynomial-time routing algorithm.

Many Cayley graphs including a burnt pancake graph with degree k have been proven to be k-connected [1]. Hence, an n-burnt pancake graph is n-connected. Therefore, it is the purpose of this study to obtain paths from an arbitrary source node to n destination nodes that are disjoint except for the source node in a polynomial time of n.

3. Algorithm

3.1 Cases by Distribution of Destination Nodes

In the case of $n = 2$, it is trivial to find the solution to the node-to-set disjoint paths problem in an n-burnt pancake graph. Hence, we assume that $n \geqq 3$ in the following discussion. Taking advantage of the symmetric property of B_n, we fix the source node to $\boldsymbol{s} = (1, 2, \cdots, n)$. Let $D = \{\boldsymbol{d}_1, \boldsymbol{d}_2, \cdots, \boldsymbol{d}_n\}$ be the set of n destination nodes. Then we consider the following two cases.

Case I All destination nodes belong to the subgraph which includes the source node ($D \subset B_{n-1}n$).

Case II There exists a destination node which does not belong to the same subgraph as the source node ($D \not\subset B_{n-1}n$).

In the following sections, we describe procedures to obtain disjoint paths from the source node to the destination nodes for these two cases.

3.2 Case I

In Case I ($D \subset B_{n-1}n$) as described in the previous section, disjoint paths are obtained by the following procedure.

1. For $B_{n-1}n$, we apply our algorithm recursively to obtain disjoint paths from $\boldsymbol{s}$ to $\{\boldsymbol{d}_1, \boldsymbol{d}_2, \cdots, \boldsymbol{d}_{n-1}\}$. If $\boldsymbol{d}_n$ is on one of these paths, say a path from $\boldsymbol{s}$ to $\boldsymbol{d}_h$, then exchange the indices of $\boldsymbol{d}_h$ and $\boldsymbol{d}_n$ and discard the subpath between $\boldsymbol{d}_h$ and $\boldsymbol{d}_n$. See Fig. 3. Large circles represent burnt pancake subgraphs while small circles represent nodes. The destination nodes are emphasized by thick lines.
2. Select edge $(\boldsymbol{d}_n, P_n(\boldsymbol{d}_n))$.
3. Select a path through the traversal class $C(\boldsymbol{s})$ starting from $\boldsymbol{s}$ to a node $\tilde{\boldsymbol{s}}$ in $B_{n-1}\overline{d}_1 \cap C(\boldsymbol{s})$, such that the path does not include any nodes

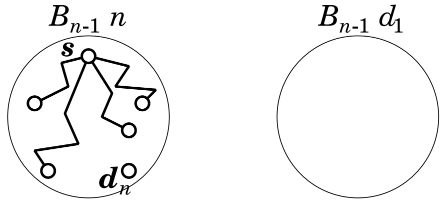

Fig. 3 Case I, Step 1.

in $B_{n-1}n$ nor $B_{n-1}\overline{d}_1$, and assume that $\boldsymbol{d}_n = (d_1, d_2, \cdots, d_n)$. See Fig. 4.

4. In $B_{n-1}\overline{d}_1$, find a path from $\tilde{\boldsymbol{s}}$ to $P_n(\boldsymbol{d}_n)$ by the polynomial-time routing algorithm. See Fig. 5.

3.3 Case II

In Case II ($D \not\subset B_{n-1}n$) as described previously, disjoint paths are obtained by the following procedure.

1. Divide the destination nodes into the following two sets D_1 and D_2.
    - D_1: The set of destination nodes such that there is a traversal class path from that destination node to one of the nodes in $B_{n-1}n$ that does not include any other destination nodes.
    - D_2: Destination nodes other than D_1.

    Here, we assume that $D_1 = \{\boldsymbol{d}_1, \boldsymbol{d}_2, \cdots, \boldsymbol{d}_k\}$ and $D_2 = \{\boldsymbol{d}_{k+1}, \boldsymbol{d}_{k+2}, \cdots, \boldsymbol{d}_n\}$ without loss of generality. See Fig. 6. Note that dashed ellipses represent node sets and horizontal dashed lines represent ring structures of traversal classes. Note also that both ends of each dashed line are assumed to be connected.
2. For each destination node $\boldsymbol{d}_i$ ($k+1 \leqq i \leqq n$) in D_2, select a neighbor node $\tilde{\boldsymbol{d}}_i$ of $\boldsymbol{d}_i$ that satisfies following conditions:
    - $C(\tilde{\boldsymbol{d}}_i) \cap D = \emptyset$;
    - $\forall j (\neq i)$, $C(\tilde{\boldsymbol{d}}_i) \neq C(\tilde{\boldsymbol{d}}_j)$

    See Fig. 7. For each node in D_2, a neighbor node is selected such that its traversal class is different from those of others. Note that these neighbor nodes can be selected based on the properties of traversal classes. That is, the number of nodes in D_2 is at most $n-2$ and each node $\boldsymbol{d}_i$ in D_2 has n neighbor nodes. Two of those neighbor nodes belong to the same traversal class as $\boldsymbol{d}_i$ and the remaining $n-2$ nodes belong to different traversal classes from $C(\boldsymbol{d}_i)$ and from each other. Therefore, we can select a node $\tilde{\boldsymbol{d}}_j$ which satisfies the above conditions. In the rest of this paper, let $\tilde{\boldsymbol{d}}_i = \boldsymbol{d}_i$ ($1 \leqq i \leqq k$).
3. For i ($1 \leqq i \leqq n$), establish a traversal class path between $\tilde{\boldsymbol{d}}_i$ and a node $\boldsymbol{c}_i$ in $B_{n-1}n$ such that the path does not include other $\tilde{\boldsymbol{d}}_j$'s nor other nodes in $B_{n-1}n$. Note that if there exists a node $\tilde{\boldsymbol{d}}_i$ such that $\tilde{\boldsymbol{d}}_i \notin B_{n-1}n$ and $\tilde{\boldsymbol{d}}_i \in C(\boldsymbol{s})$, then select a path such that $\boldsymbol{s}$ is included in $\{\boldsymbol{c}_1, \boldsymbol{c}_2, \cdots, \boldsymbol{c}_n\}$. Moreover, note that if $\tilde{\boldsymbol{d}}_i$ is in $B_{n-1}n$ then the path consists of the node itself. In addition, select edges $(\tilde{\boldsymbol{d}}_i, \boldsymbol{d}_i)$ ($k+1 \leqq i \leqq n$). See Fig. 8.
4. If there exists an i^* such that $\boldsymbol{s} \in C(\tilde{\boldsymbol{d}}_{i^*})$, apply our algorithm recursively on $B_{n-1}n$ to obtain disjoint paths from $\boldsymbol{s}$ to $\tilde{\boldsymbol{d}}_i$ ($i \neq i^*$) and terminate. See Fig. 9.
5. Select a node which is on one of paths obtained in Step 3 and not included in $B_{n-1}n$. Let $\boldsymbol{u}$ represent the node. Assume that $\boldsymbol{u} \in B_{n-1}h$. Select a traversal class path between $\boldsymbol{s}$ and a node $\tilde{\boldsymbol{s}}$ in $B_{n-1}h$ such that the path does not include other nodes in $B_{n-1}n$ nor $B_{n-1}h$. See Fig. 10. A subpath between $\boldsymbol{s}$ and $\tilde{\boldsymbol{s}}$ is selected. Note that it is a

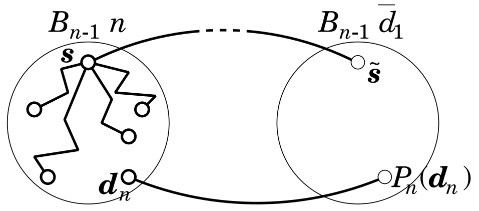

Fig. 4 Case I, Step 3.

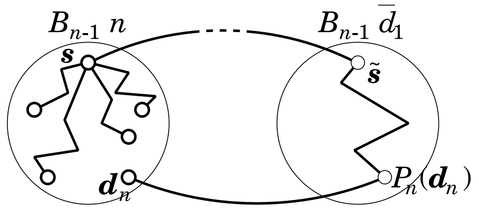

Fig. 5 Case I, Step 4.

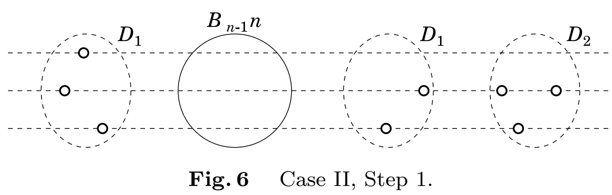

Fig. 6 Case II, Step 1.

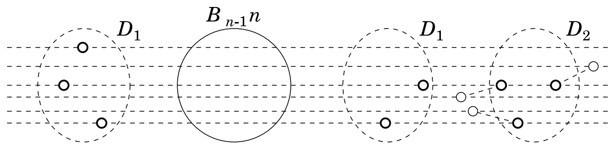

Fig. 7 Case II, Step 2.

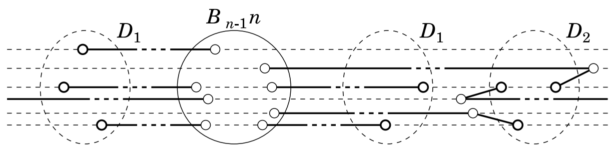

Fig. 8 Case II, Step 3.

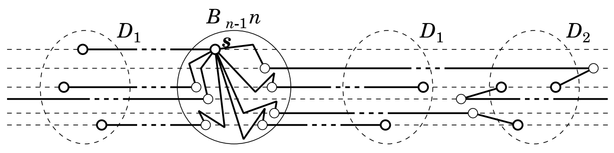

Fig. 9 Case II, Step 4.

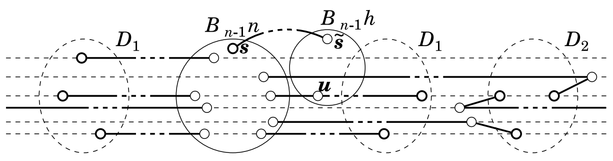

Fig. 10 Case II, Step 5.

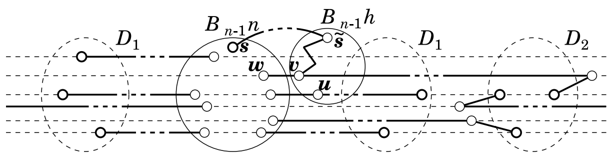

Fig. 11 Case II, Step 6.

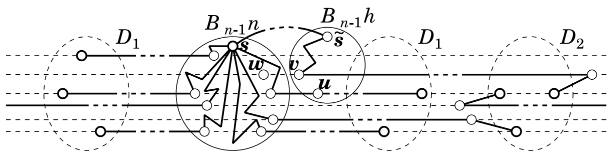

Fig. 12 Case II, Step 7.

traversal class path though it is not horizontal.

6. Find a path from $\tilde{s}$ to $\boldsymbol{u}$ in $B_{n-1}h$ by the polynomial-time routing algorithm. If this path contains a node (or nodes) on the path obtained in Step 3, let $\boldsymbol{v}$ denote the node and discard the subpath from node $\boldsymbol{v}$ to node $\boldsymbol{u}$. See Fig. 11. Otherwise, let $\boldsymbol{u}$ represent node $\boldsymbol{v}$.
7. Let $\boldsymbol{w}$ be the terminal node in $B_{n-1}n$ of the path which was obtained in Step 3 and includes $\boldsymbol{v}$. Discard the subpath from $\boldsymbol{w}$ to $\boldsymbol{v}$. Moreover, apply our algorithm recursively to $B_{n-1}n$ to obtain disjoint paths from $\boldsymbol{s}$ to $\{\boldsymbol{c}_1, \boldsymbol{c}_2, \cdots, \boldsymbol{c}_n\} - \{\boldsymbol{w}\}$ and terminate. See Fig. 12.

4. Proof of Correctness and Estimation of Complexities

In this chapter, we give a proof of the correctness of our algorithm and estimate its time complexity and the sum of path lengths.

Theorem 2: Paths generated by our algorithm are disjoint except for the source node. Let $T(n)$ and $L(n)$ represent the time complexity and the complexity of the sum of path lengths for an n-burnt pancake graph, respectively. Then $T(n) = O(n^5)$ and $L(n) = O(n^3)$.

(Proof) It can be proved from induction on n and the following two lemmas.

Lemma 1: The paths generated by the procedure for Case I are disjoint except for the source node. The time complexity of this procedure is $T(n-1) + O(n^4)$ and the sum of path lengths is $L(n-1) + O(n)$.

Table 2 Time complexity and sum of path lengths for each step in the procedure for Case II.

	Time Complexity	Sum of Path Lengths
Step 1	$O(n^3)$	—
Step 2	$O(n^4)$	—
Step 3	$O(n^3)$	$O(n^2)$
Step 4	$T(n-1) + O(n^2)$	$L(n-1)$
Step 5	$O(n^2)$	$O(n)$
Step 6	$O(n^3)$	$O(n)$
Step 7	$T(n-1)$	$L(n-1)$

(Proof) All paths obtained in Step 1 are known to be disjoint except for the source node $\boldsymbol{s}$ by the induction hypothesis. Moreover, the path from $\boldsymbol{s}$ to $\boldsymbol{d}_n$ generated by Steps 2 to 4 is outside of $B_{n-1}n$ except for $\boldsymbol{s}$ and $\boldsymbol{d}_n$. In addition, all paths inside $B_{n-1}n$ are constructed to not include $\boldsymbol{d}_n$. Therefore, all paths generated by this procedure are disjoint except for $\boldsymbol{s}$.

The time complexity in Step 1 is $T(n-1) + L(n-1) \times n$. It is equal to $T(n-1) + O(n^4)$ by the induction hypothesis. In addition, the sum of path lengths is $L(n-1)$. The time complexity of Steps 2 to 4 is $O(n^2)$ and the sum of path lengths is $O(n)$. Hence, the total time complexity and the sum of path lengths are $T(n-1) + O(n^4)$ and $L(n-1) + O(n)$, respectively.

Lemma 2: The paths generated by the procedure for Case II are disjoint except for the source node. The time complexity of this procedure is $T(n-1) + O(n^4)$ and the sum of path lengths is $L(n-1) + O(n^2)$.

(Proof) Paths generated in Step 3 are separate traversal class paths that start from different nodes $\boldsymbol{c}_i$ in $B_{n-1}n$ and end in different destination nodes $\boldsymbol{d}_i$ or their neighbor nodes $\tilde{\boldsymbol{d}}_i$ with edges $(\tilde{\boldsymbol{d}}_i, \boldsymbol{d}_i)$. Hence, they are all disjoint. The paths obtained in Step 4 are disjoint except for $\boldsymbol{s}$ by induction hypothesis. They are trivially disjoint from the paths obtained in Step 3 except for $\boldsymbol{c}_i$. Paths obtained in Step 6 are necessarily disjoint from other paths except for $\boldsymbol{s}$ and $\boldsymbol{v}$ because of the manner of their construction. In addition, paths obtained by the recursive application of our algorithm are disjoint from each other except for $\boldsymbol{s}$ by induction hypothesis, and also disjoint from other paths except for $\boldsymbol{c}_i$. Therefore, the paths generated by this procedure are all disjoint except for the source node $\boldsymbol{s}$.

The time complexity and the sum of path lengths for each step are summarized in Table 2.

Table 2 shows the time complexity of this procedure to be $T(n-1)+O(n^4)$, and the sum of path lengths to be $L(n-1) + O(n^2)$.

5. Computer Simulation

To evaluate the average performance of our algorithm, we conducted the following computer simulation for an n-burnt pancake graph. Our algorithm is implemented in the functional programming language Haskell. The

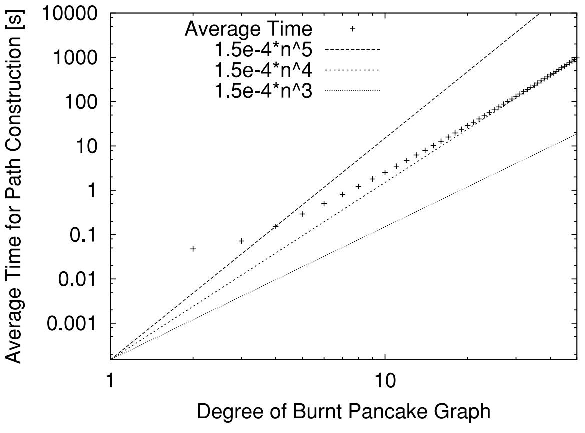

Fig. 13 Average time for path construction.

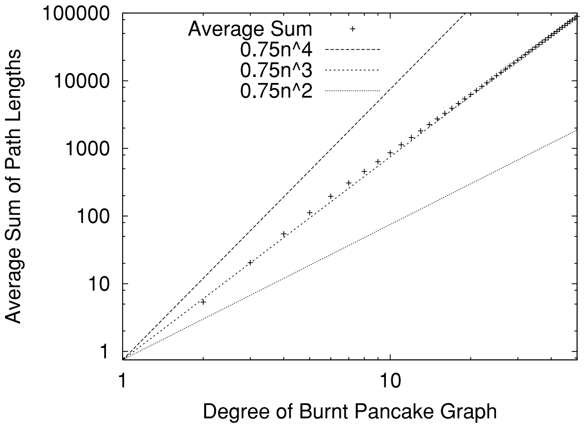

Fig. 14 Average sum of paths length.

program is compiled with the Glasgow Haskell Compiler with options `-O` and `-fglasgow-exts` and is executed on a target machine equipped with an Intel Pentium III 700 MHz CPU and a 512 MB memory unit.

1. Select a source node $\boldsymbol{s}$ randomly.
2. Select n different destination nodes $\boldsymbol{d}_1, \boldsymbol{d}_2, \cdots, \boldsymbol{d}_n$ other than $\boldsymbol{s}$ randomly.
3. Apply our algorithm and measure the execution time and the sum of path lengths.

The simulation was performed 1,000 times for each n of $n = 2, \cdots, 50$. Figures 13 and 14 show the average execution time and the average sum of path lengths, respectively. Three lines in each figure are added to estimate the decline of plotted data. From these figures, the average execution time and the average sum of path lengths are both in polynomial order of n, and about $O(n^4)$ and $O(n^3)$, respectively.

6. Conclusion

In this study, we selected an n-burnt pancake graph as a target, and we have proposed an algorithm which finds n disjoint paths for one source node and n destination nodes in polynomial-order time of the degree n. In addition, we have given a proof of the correctness of our algorithm and we have shown that the time complexity and the complexity of the sum of path lengths are $O(n^5)$ and $O(n^3)$, respectively. Moreover, we have evaluated the average performance of our algorithm by computer simulation and have shown that the average time complexity and the average complexity of sum of path lengths are $O(n^4)$ and $O(n^3)$, respectively.

Future works include improvement of our algorithm to find shorter disjoint paths in a shorter time and the application of this algorithm to other topologies such as generalized pancake graphs [12]. Verification of our algorithm in practical use is also a future work.

References

[1] S.B. Akers and B. Krishnamurthy, "On group graphs and their fault tolerance," IEEE Trans. Comput., vol.C-36, no.7, pp.885–888, July 1987.

[2] 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.

[3] 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.

[4] 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.

[5] 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, May 1992.

[6] 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.

[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] W.H. Gates and C.H. Papadimitriou, "Bounds for sorting by prefix reversal," Discrete Mathematics, vol.27, pp.47–57, 1979.

[9] 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.

[10] 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.

[11] M. Heydari and I.H. Sudborough, "On sorting by prefix reversals and the diameter of pancake networks," Lecture Notes in Computer Science, vol.678, pp.218–227, 1993.

[12] M.P. Justan, F.P. Muga II, and I.H. Sudborough, "On the generalization of the pancake network," Proc. 2002 Int'l Symp. Parallel Architectures, Algorithms, and Networks, pp.173–178, Dec. 2002.

[13] 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.

[14] 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.

[15] S. Madhavapeddy and I.H. Sudborough, "A topological property of hypercubes: Node disjoint paths," Proc. 2nd IEEE Symp. Parallel and Distributed Processing, pp.532–539, 1990.

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

[17] C.L. Seitz, "The cosmic cube," Commun. ACM, vol.28, no.7, pp.22–33, July 1985.

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.