IEICE TRANS. INF. & SYST., VOL.E86–D, NO.9 SEPTEMBER 2003

PAPER Special Issue on Parallel and Distributed Computing, Applications and Technologies

Node-to-Set Disjoint Paths Problem in Pancake Graphs

Keiichi KANEKO[†], ***Regular Member and*** **Yasuto SUZUKI**[†], ***Student Member***

SUMMARY In this paper, we give an algorithm for the node-to-set disjoint paths problem in pancake graphs with its evaluation results. The algorithm is of polynomial order of n for an n-pancake graph. It is based on recursion and divided into two cases according to the distribution of destination nodes in classes into which all the nodes in a pancake graph are categorized. The sum of lengths of paths obtained and the time complexity of the algorithm are estimated and the average performance is evaluated based on computer simulation.

key words: *interconnection networks, graph algorithms, pancake graph, node-to-set disjoint paths, parallel computing*

1. Introduction

Recently, research in parallel and distributed computation has become more significant because we cannot expect drastic improvement of performance in sequential computation in the future. Moreover, extensive research on so-called massively parallel machines has been conducted in recent years. Hence, many complex topologies of interconnection networks [1], [6] have been proposed to replace simple networks such as a hypercube and a mesh. A pancake graph [1] is a new topology that shows promise in that it has a low degree and a small diameter relative to the number of nodes. Table 1 shows a comparison of an n-pancake graph Π_n with an n-rotator graph R_n, an n-star graph S_n, an n-cube Q_n, an (n,k)-de Bruijn graph $B_{n,k}$, and an (n,k)-Kautz graph $K_{n,k}$. From this table, we can see that the n-pancake graph affords better performance than do the topologies S_n and Q_n as it can connect more nodes for a given diameter or a given degree. Though the n-pancake graph is inferior to the n-rotator graph, the (n,k)-de Bruijn graph and the (n,k)-Kautz graph in this respect, the n-rotator graph is a directed graph that is unsuitable for applications which require extensive local communications, and the latter two topologies have neither symmetry nor recursive structure, so they are impractical for parallel executions of some applications.

Table 1 Comparison of a pancake graph with other topologies.

	# nodes	degree	diameter
Π_n	$n!$	$n-1$	$\leqq 5(n+1)/3$
R_n	$n!$	$n-1$	$n-1$
S_n	$n!$	$n-1$	$\lfloor \frac{3}{2}(n-1) \rfloor$
Q_n	2^n	n	n
$B_{n,k}$	n^k	$2n$	k
$K_{n,k}$	$n^k + n^{k-1}$	n	k

Manuscript received November 20, 2002.
Manuscript revised March 14, 2003.

[†]The authors are with the Faculty of Technology, Tokyo University of Agriculture and Technology, Koganei-shi, 184–8588 Japan.

Unfortunately, there still remain unknowns in several metrics for this topology despite intense research activity [2]–[5], [7], [12]. Among the unsolved problems is the node-to-set disjoint paths problem: Given a source node $\boldsymbol{s}$ and a set $D = \{\boldsymbol{d}_1, \boldsymbol{d}_2, \cdots, \boldsymbol{d}_k\}$ ($\boldsymbol{s} \notin D$) of k destination nodes in a k-connected graph $G = (V, E)$, find k paths from $\boldsymbol{s}$ to $\boldsymbol{d}_i$ ($1 \leqq i \leqq k$) which are node-disjoint except for $\boldsymbol{s}$. This is one of the most important issues in the design and implementation of parallel and distributed computing systems [8], [10], [13] as is the node-to-node disjoint paths problem [9], [11]. Once these k paths are obtained, they achieve fault tolerance; that is, at least one path can survive with $k-1$ faulty components.

In general, node-disjoint paths can be obtained in polynomial order time of $|V|$ by making use of the maximum flow algorithm. However, in an n-pancake graph, the number of nodes is equal to $n!$, so in this case its complexity is too large. In this paper, we propose an algorithm which is of polynomial order of n instead of $n!$ and present the results of computer simulation.

The rest of this paper is organized as follows. Section 2 introduces pancake graphs as well as the notion of classes, and the problem is formalized. Section 3 explains our algorithm in detail. Computer simulation is reported in Sect. 4. Section 5 describes conclusions and future work.

2. Preliminaries

Definition 1: For an arbitrary permutation $\boldsymbol{u} = (a_1, a_2, \cdots, a_n)$ of n symbols, $1, 2, \cdots, n$, the prefix reversal operation $P_i(\boldsymbol{u})$ ($2 \leqq i \leqq n$) is defined as follows: $P_i(\boldsymbol{u}) = (a_i, a_{i-1}, \cdots, a_1, a_{i+1}, a_{i+2}, \cdots, a_n)$.

Definition 2: An n-pancake graph, Π_n, has $n!$ nodes. Each node has a unique address which is a permutation of n symbols $1, 2, \cdots, n$. A node which has an address $\boldsymbol{u} = (a_1, a_2, \cdots, a_n)$ is adjacent to $n-1$ nodes whose addresses are elements of the set $\{P_i(\boldsymbol{u}) \mid 2 \leqq i \leqq n\}$.

Figure 1 shows an example of a 4-pancake graph. Note that the address $(a_1, a_2, \cdots, a_n)$ is denoted as

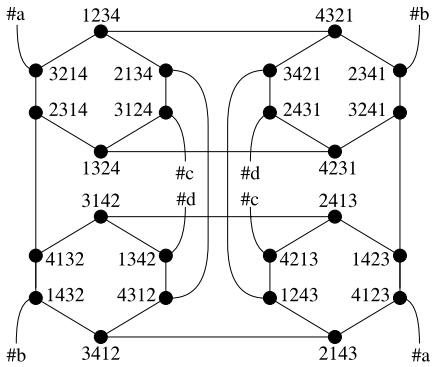

Fig. 1 An example of a 4-pancake graph.

$a_1a_2\cdots a_n$ in the figure and that a pair of edges ending in the same letter, e.g. 'a', are connected.

Definition 3: In an n-pancake graph, a subgraph which is induced by nodes that have a common symbol k at the last position of their addresses constitutes an $(n-1)$-pancake graph. This subpancake graph is specified by $\Pi_{n-1}k$ using the common symbol k as an index.

Definition 4: For an arbitrary node $\boldsymbol{u}$ in an n-pancake graph Π_n, a subset of nodes obtained by alternative applications of P_n and P_{n-1} is a class to which $\boldsymbol{u}$ belongs and is denoted by $C(\boldsymbol{u})$.

Theorem 1: With respect to classes in an n-pancake graph where $n \geqq 3$, the following properties hold:

1. Each node belongs to exactly one class.
2. Each class has $2n$ nodes which together constitute a ring structure.
3. Each subpancake graph overlaps with each class by exactly two nodes.
4. Each node has two neighbor nodes that belong to the same class as the node and $n-3$ other neighbor nodes which belong to different classes from each other.

Proof: Each property is proved as follows:

1. For any node $\boldsymbol{u}$ in Π_n, assume that $\boldsymbol{u}$ belongs to classes C_1 and C_2. Let $\boldsymbol{a}$ and $\boldsymbol{b}$ be nodes in C_1 and C_2, respectively. Then there exists a path from $\boldsymbol{a}$ to $\boldsymbol{b}$ that is obtained by alternative application of P_n and P_{n-1} to $\boldsymbol{a}$. Hence, $\boldsymbol{b} \in C_1$ and $C_2 \subset C_1$. Similarly, $C_1 \subset C_2$ holds and $C_1 = C_2$.
2. For any node $\boldsymbol{u} = (u_1, u_2, \cdots, u_n)$ in Π_n, $P_n(\boldsymbol{u}) = (u_n, u_{n-1}, \cdots, u_1)$, $P_{n-1}(P_n(\boldsymbol{u})) = (u_2, u_3, \cdots, u_n, u_1), \cdots$, $P_n \circ (P_{n-1} \circ P_n)^{k-1}(\boldsymbol{u}) = (u_{k-1}, u_{k-2}, \cdots, u_1, u_n, u_{n-1}, \cdots, u_k)$, $(P_{n-1} \circ P_n)^k(\boldsymbol{u}) = (u_{k+1}, u_{k+2}, \cdots, u_n, u_1, u_2, \cdots, u_k), \cdots$, $P_n \circ (P_{n-1} \circ P_n)^{n-1}(\boldsymbol{u}) = (u_{n-1}, u_{n-2}, \cdots, u_1, u_n)$, $(P_{n-1} \circ P_n)^n(\boldsymbol{u}) = (u_1, u_2, \cdots, u_n) = \boldsymbol{u}$. Focusing on the first and the last symbols of the address of each node, these $2n$ nodes are all distinct and they constitute a ring structure.

```
function route(s=(s_1,...,s_n),d=(d_1,...,d_n));
begin
  E := ∅;
  for i := n to 1 step -1
    if s_i ≠ d_i then begin
      find k such that s_k = d_i;
      s' := P_k(s);
      if k > 1 then E := E ∪ {(s,s')};
      s := P_i(s');
      E := E ∪ {(s',s)}
    end;
  return E
end
```

Fig. 2 A polynomial time routing algorithm.

3. The proof of the previous property shows that, for any subpancake graph $\Pi_{n-1}k$, each class contains exactly two nodes whose addresses have symbol k at their last position.
4. For each node $\boldsymbol{u} = (u_1, u_2, \cdots, u_n)$ in Π_n, $P_n(\boldsymbol{u})$ and $P_{n-1}(\boldsymbol{u})$ belong to $C(\boldsymbol{u})$ by definition. Other neighbor nodes $P_i(\boldsymbol{u})$ $(2 \leqq i \leqq n-2)$ are in the same subpancake as $\boldsymbol{u}$. Hence, from the previous property, they belong to different classes from $C(\boldsymbol{u})$. In addition, for any pair of nodes $P_h(\boldsymbol{u})$ and $P_k(\boldsymbol{u})$ $(2 \leqq h < k \leqq n-1)$, $P_{n-1}(P_h(\boldsymbol{u})) = (u_{n-1}, u_{n-2}, \cdots, u_{h+1}, u_1, u_2, \cdots, u_h, u_n)$ and $P_k(\boldsymbol{u}) = (u_k, u_{k-1}, \cdots, u_1, u_{k+1}, \cdots, u_n)$. Then, there is no solution for h and k $(2 \leqq h < k \leqq n-1)$ which satisfy the equality of these nodes, Hence, $C(P_h(\boldsymbol{u})) \neq C(P_k(\boldsymbol{u}))$.

For instance, there are three classes, C_1, C_2 and C_3, each of which consists of 8 nodes, in a 4-pancake graph Π_4.

$$\begin{aligned} C_1 = \{&(1,2,3,4),(4,3,2,1),(2,3,4,1),(1,4,3,2),\\ &(3,4,1,2),(2,1,4,3),(4,1,2,3),(3,2,1,4)\}\\ C_2 = \{&(2,1,3,4),(4,3,1,2),(1,3,4,2),(2,4,3,1),\\ &(3,4,2,1),(1,2,4,3),(4,2,1,3),(3,1,2,4)\}\\ C_3 = \{&(1,3,2,4),(4,2,3,1),(3,2,4,1),(1,4,2,3),\\ &(2,4,1,3),(3,1,4,2),(4,1,3,2),(2,3,1,4)\} \end{aligned}$$

In addition, a traversal from the beginning in each class shows that the nodes in it constitute a ring structure.

Definition 5: In a pancake graph, a class path is a subpath of a ring structure constituted by a class.

Finally, we present a simple unicast routing algorithm because no shortest-path-routing algorithm of polynomial order of n for n-pancake graphs was found. Figure 2 shows the algorithm. Its time complexity is $O(n^2)$ and its path length is $O(n)$.

3. Algorithm NS

In this section, we propose an algorithm called NS for

the node-to-set disjoint paths problem in an n-pancake graph.

Algorithm NS consists of Procedures 1 and 2. Procedure 1 concerns the special case where all destinations belong to the subpancake that includes the source node. It applies Algorithm NS within the subpancake to obtain $n-2$ disjoint paths and then establishes a path between the remaining destination node and the source via nodes outside of the subpancake. On the other hand, Procedure 2 concerns a general case. It first constructs $n-1$ disjoint paths from $n-1$ nodes in the subpancake graph that includes the source to the destination nodes by making use of class paths. Then Procedure 2 reconnects one of these paths to the source by discarding its subpath. Finally it applies Algorithm NS within the subpancake recursively to complete $n-2$ other disjoint paths. Each procedure invokes Algorithm NS within itself. Hence, Algorithm NS has a recursive structure.

3.1 Classification

If $n \leqq 2$, the problem is trivial. That is, a 2-pancake graph consists of two nodes and an edge between them. Hence, if one node is the source, then the other one is the destination, and the path is the edge itself. This is the base case of Algorithm NS. Therefore, we assume $n \geqq 3$ in the following. We can fix the source node as $\boldsymbol{s} = (1, 2, \cdots, n)$, taking advantage of the symmetric properties of Π_n. Let $D = \{\boldsymbol{d}_1, \boldsymbol{d}_2, \cdots, \boldsymbol{d}_{n-1}\}$ be the set of destination nodes and $\mathcal{C} = \{C_1, C_2, \cdots, C_k\}$ be the collection of classes to which all destination nodes belong. Then let us consider the following two cases.

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

Case 2 There exists a destination node in a subpancake graph to which the source node does not belong. ($D \not\subset \Pi_{n-1}n$).

3.2 Procedure 1

In this section, we present Procedure 1 to address **Case 1** in which $D \subset \Pi_{n-1}n$. This procedure is used to construct $n-1$ paths from the source node $\boldsymbol{s}$ to the set of destination nodes $D = \{\boldsymbol{d}_1, \boldsymbol{d}_2, \cdots, \boldsymbol{d}_{n-1}\}$ which are node-disjoint except for the source. Note that, when Algorithm NS is applied recursively within a subpancake graph in Procedure 1, the distribution of the revised set of destination nodes with respect to the subpancake graph is used to determine whether to apply Procedure 1 or Procedure 2.

Step 1 Apply Algorithm NS recursively within $\Pi_{n-1}n$ to obtain $n-2$ paths from $\boldsymbol{s}$ to $D - \{\boldsymbol{d}_{n-1}\}$ which are disjoint except for $\boldsymbol{s}$. If $\boldsymbol{d}_{n-1}$ is on one of these paths, say, a path from $\boldsymbol{s}$ to $\boldsymbol{d}_h$, then discard the subpath from $\boldsymbol{d}_{n-1}$ to $\boldsymbol{d}_h$ and exchange the indices of $\boldsymbol{d}_{n-1}$ and $\boldsymbol{d}_h$. See Fig. 3.

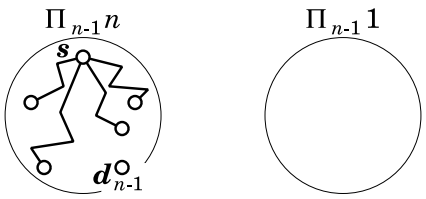

Fig. 3 Recursive application of Algorithm NS.

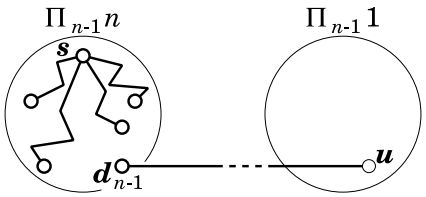

Fig. 4 Construction of a path through class $C(\boldsymbol{d}_{n-1})$.

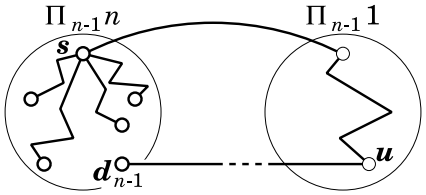

Fig. 5 Construction of a path from $\boldsymbol{s}$ to $\boldsymbol{u}$.

Step 2 Construct a class path from node $\boldsymbol{d}_{n-1}$ to the nearest node $\boldsymbol{u}$ in $C(\boldsymbol{d}_{n-1}) \cap \Pi_{n-1}1$ such that node $P_{n-1}(\boldsymbol{d}_{n-1})$ is not included in the path. See Fig. 4.

Step 3 Select an edge between $\boldsymbol{s}$ and $P_n(\boldsymbol{s})$. Construct a path from $P_n(\boldsymbol{s})$ to $\boldsymbol{u}$ by using the polynomial time routing algorithm shown in Fig. 2. See Fig. 5.

3.3 Procedure 2

In this section, we present Procedure 2 to address **Case 2** in which $D \not\subset \Pi_{n-1}n$. This procedure is used to construct $n-1$ paths from the source node $\boldsymbol{s}$ to the set of destinations $D = \{\boldsymbol{d}_1, \boldsymbol{d}_2, \cdots, \boldsymbol{d}_{n-1}\}$ which are node-disjoint except for $\boldsymbol{s}$. Note that, when Algorithm NS is applied recursively within a subpancake graph in Procedure 2, the distribution of the revised set of destination nodes with respect to the subpancake graph is used to determine whether to apply Procedure 1 or Procedure 2.

Step 1 For each C_i, do the following. If $|C_i \cap D| = 1$, select a node in $C_i \cap \Pi_{n-1}n$ and construct a class path from the node to the destination node in C_i such that the path does not include another node in $C_i \cap \Pi_{n-1}n$. If $|C_i \cap D| \geqq 2$, select both nodes in $C_i \cap \Pi_{n-1}n$ and construct two node-disjoint class paths from the nodes to the nearest destination

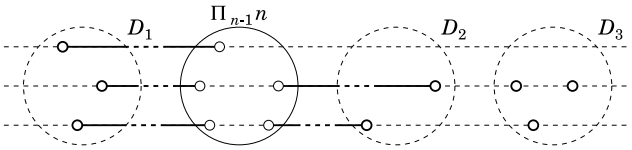

Fig. 6 Construction of paths through classes.

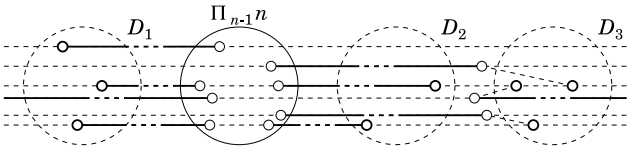

Fig. 7 Construction of paths to neighbor nodes of D_3.

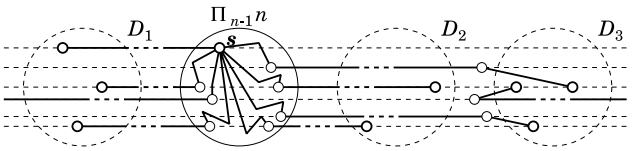

Fig. 8 Recursive application of Algorithm NS.

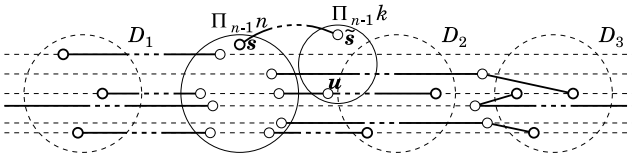

Fig. 9 Specification of the target subpancake graph.

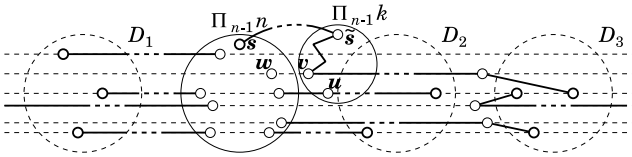

Fig. 10 Intercepting one of paths.

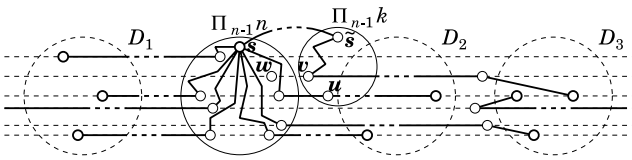

Fig. 11 Recursive application of Algorithm NS.

nodes in C_i. If there is exactly one destination node in $C(\boldsymbol{s}) - \Pi_{n-1}n$, then select a path that starts from $\boldsymbol{s}$. Let D_3 represent the set of the destination nodes that are not terminal nodes of any of the paths selected so far. In addition, the node set $D - D_3$ is divided into two subsets D_1 and D_2 so that any pair of nodes that belong to the same class is separated. See Fig. 6.

Step 2 For each destination node $\boldsymbol{d}_i$ in D_3, find its neighbor node $\boldsymbol{c}_i$ which satisfies the conditions '$C(\boldsymbol{c}_i) \notin \mathcal{C}$' and '$C(\boldsymbol{c}_i) \neq C(\boldsymbol{c}_j)$, if $i \neq j$' in a greedy manner. Note that such neighbor nodes can be selected using the fact that $|D_3| \leq n - 2k - 1$ and using Property 4 of classes.

Step 3 For each $\boldsymbol{c}_i$ obtained in Step 2, construct a class path from a node in $\Pi_{n-1}n \cap C(\boldsymbol{c}_i)$ to $\boldsymbol{c}_i$ such that each path must include only one node in $\Pi_{n-1}n \cap C(\boldsymbol{c}_i)$. If there exists a node $\boldsymbol{c}_i$ which belongs to $C(\boldsymbol{s})$, then the path that starts from $\boldsymbol{s}$ is selected. See Fig. 7.

Step 4 Select an edge between each $\boldsymbol{c}_i$ and corresponding $\boldsymbol{d}_i$.

Step 5 Among the paths selected in the previous steps, if there is one that has $\boldsymbol{s}$ as its terminal node, then apply Algorithm NS recursively to obtain $n - 2$ paths from $\boldsymbol{s}$ to terminal nodes in $\Pi_{n-1}n$ other than $\boldsymbol{s}$ which are disjoint except for $\boldsymbol{s}$ and terminate. See Fig. 8. Otherwise, go to Step 6.

Step 6 Find a subpancake that includes at least one node (not necessarily a destination) on the paths obtained in Steps 1 to 4. Let $\Pi_{n-1}k$ and $\boldsymbol{u}$ represent the subpancake and the node on the path, respectively. Then construct a class path from $\boldsymbol{s}$ to the nearest node $\tilde{\boldsymbol{s}}$ in $\Pi_{n-1}k \cap C(\boldsymbol{s})$ such that the path does not include $P_{n-1}(\boldsymbol{s})$. See Fig. 9.

Step 7 Construct a path from $\tilde{\boldsymbol{s}}$ to $\boldsymbol{u}$ by using the algorithm shown in Fig. 2. If the path includes some nodes on the paths selected in Steps 1 to 4, let $\boldsymbol{v}$ represent the nearest such node from $\tilde{\boldsymbol{s}}$ and discard the subpath from $\boldsymbol{v}$ to $\boldsymbol{u}$. Otherwise, let $\boldsymbol{u}$ be $\boldsymbol{v}$. Moreover, let $\boldsymbol{w}$ be the terminal node in $\Pi_{n-1}n$ of the path selected in a previous step that includes node $\boldsymbol{v}$. Then discard the subpath from $\boldsymbol{w}$ to $\boldsymbol{v}$. See Fig. 10.

Step 8 For the terminal nodes of paths selected in Steps 1 to 4 other than node $\boldsymbol{w}$, apply Algorithm NS recursively to obtain $n - 2$ paths from $\boldsymbol{s}$ to these nodes which are disjoint except for $\boldsymbol{s}$, and terminate. See Fig. 11.

4. Proof of Correctness and Estimation of Complexities

In this section, we give proofs that Algorithm NS shown in the previous section constructs $n - 1$ paths from $\boldsymbol{s}$ to the destination nodes which are node-disjoint except for $\boldsymbol{s}$.

Theorem 2: $n - 1$ paths constructed by Algorithm NS are node-disjoint except for $\boldsymbol{s}$. For n-pancake graph, the time complexity $T(n)$ and the sum of the path lengths $L(n)$ are $O(n^5)$ and $O(n^3)$, respectively.

Proof: By hypothesis of induction on n, the following two lemmas are proved. They imply the theorem immediately.

Lemma 1: The $n-1$ paths obtained by Procedure 1 are node-disjoint except for $\boldsymbol{s}$. The time complexity and the sum of path lengths are $T(n-1)+O(n^4)$ and $L(n-1)+O(n)$, respectively.

Proof: All paths constructed in Step 1 are node-disjoint except for $\boldsymbol{s}$ by induction hypothesis. These paths do not contain $\boldsymbol{d}_{n-1}$. Except for $\boldsymbol{s}$ and $\boldsymbol{d}_{n-1}$, all nodes on the path constructed in Steps 2 and 3 are outside of $\Pi_{n-1}n$. So, these $n-1$ paths are node-disjoint except for $\boldsymbol{s}$.

The time complexity of Step 1 is $T(n-1)+L(n-1)\times n$. By induction hypothesis, it is equivalent to $T(n-1)+O(n^4)$. And the sum of path lengths of Step 1 is $L(n-1)$. For Steps 2 and 3, the time complexity and the sum of path lengths are $O(n^2)$ and $O(n)$, respectively. Hence, the total complexities of the time and the sum of path lengths are $T(n-1)+O(n^4)$ and $L(n-1)+O(n)$, respectively.

Lemma 2: The $n-1$ paths obtained by Procedure 2 are node-disjoint except for $\boldsymbol{s}$. The time complexity and the sum of path lengths are $T(n-1)+O(n^4)$ and $L(n-1)+O(n^2)$, respectively.

Proof: All paths selected in Steps 1 and 3 are node-disjoint, because of the disjointness of classes. These paths and edges selected in Step 4 are obviously disjoint except for their joint nodes. Paths obtained in Steps 5 and 8 are node-disjoint except for $\boldsymbol{s}$ by induction hypothesis. Similar discussion holds for other cases. Hence, all paths constructed in Procedure 2 are node-disjoint except for $\boldsymbol{s}$.

The time complexities of Steps 5 and 8 are both $T(n-1)$. The time complexity of Step 2 is $O(n^4)$, which governs the case. The sums of path lengths obtained in Step 5 and 8 are both $L(n-1)$. Those of Steps 1 and 3 are both $O(n^2)$, which govern the case. Then, in total, the time complexity and the sum of path lengths are of $T(n-1)+O(n^4)$ and $L(n-1)+O(n^2)$, respectively.

5. Computer Simulation

To evaluate average performance of Algorithm NS, we repeated the following procedures for random combinations of target nodes where the source node $\boldsymbol{s}$ is fixed to the identity permutation $(1,2,\cdots,n)$. The program is written in the functional language Haskell, and it is compiled by the Glasgow Haskell Compiler `ghc` with `-O` and `-fglasgow-exts` options.

1. Set $n-1$ destination nodes other than $\boldsymbol{s}$ randomly.
2. Invoke Algorithm NS, count the sum of path lengths, and measure the execution time.

We show the results of the average execution time and the average sum of path lengths in Figs. 12 and 13, respectively. They are obtained by 10,000 iterations for each n. The horizontal axes represent the number expressed by n in both figures, while the vertical axes in Figs. 12 and 13 represent the average execution time in seconds and the average sum of path lengths obtained by Algorithm NS, respectively.

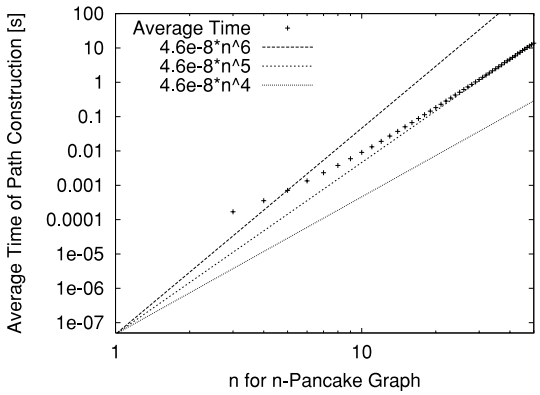

Fig. 12 Average execution time of Algorithm NS.

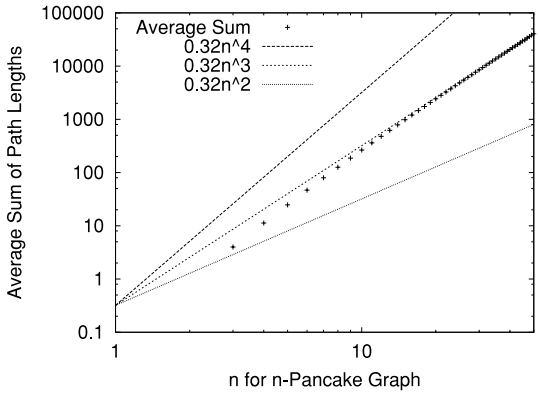

Fig. 13 Average sum of path lengths obtained by Algorithm NS.

These figures show that Algorithm NS results in $n-1$ node-disjoint paths, the sum of whose length is $O(n^3)$, and execution time is $O(n^5)$.

6. Conclusions

In this paper, we proposed a polynomial algorithm for the node-to-set disjoint paths problem in n-pancake graphs whose time complexity is $O(n^5)$ and the sum of path lengths is $O(n^3)$. We also conducted computer simulation and showed that the average execution time and the average sum of path lengths are $O(n^5)$ and $O(n^3)$, respectively. Future work includes improvement of the algorithm to generate shorter paths in shorter execution time as well as application of our approach to other topologies of interconnection networks.

Acknowledgement

We really appreciate the comments and suggestions from anonymous reviewers. This work was partly supported by Grant-in-Aid for Scientific Research (C) of

JSPS under Grant No. 13680398 and Grant-in-Aid for JSPS Fellows.

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 Conf. Vector and Parallel Processing, pp.565–570, Lyon, Sept. 1992.
[3] S.G. Akl and K. Qiu, "A novel routing scheme on the star and pancake interconnection networks and its applications," Parallel Computing, 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, May 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.477–782, 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-node disjoint paths problem in a pancake graph," Proc. Int'l Conf. Software Engineering, Artificial Intelligence, Networking & Parallel/Distributed Computing, pp.572–579, Nagoya, Aug. 2001.
[12] 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, Ottawa, May 1991.
[13] 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.

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 IEEE, ACM, IPSJ and JSSST.

Yasuto Suzuki is a student of Division of Electronics and Information Engineering, Graduate School of Technology, Tokyo University of Agriculture and Technology. He received the B.E. and M.E. degrees from Tokyo University of Agriculture and Technology in 2001 and 2003, respectively. His research interests include graph and network theories and fault-tolerant systems.