IEICE TRANS. INF. & SYST., VOL.E89–D, NO.10 OCTOBER 2006

PAPER

Node-Disjoint Paths Algorithm in a Transposition Graph

Yasuto SUZUKI[†], *Nonmember*, **Keiichi KANEKO**[††a)], *and* **Mario NAKAMORI**[††], *Members*

SUMMARY In this paper, we give an algorithm for the node-to-set disjoint paths problem in a transposition graph. The algorithm is of polynomial order of n for an n-transposition graph. It is based on recursion and divided into two cases according to the distribution of destination nodes. The maximum length of each path and the time complexity of the algorithm are estimated theoretically to be $O(n^7)$ and $3n-5$, respectively, and the average performance is evaluated based on computer experiments.

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

1. Introduction

In practical use of parallel and distributed computing systems, finding disjoint paths in interconnection networks is one of the fundamental issues [3], [5]–[7], [9]–[11]. Amongst them 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, find k paths from $\boldsymbol{s}$ to $\boldsymbol{d}_i$ ($1 \le i \le k$) that are node-disjoint except for $\boldsymbol{s}$. Once these k paths are obtained, they achieve fault tolerance; that is, at least one path can survive with $k-1$ faulty components. This problem can be solved by using the maximum flow technique, which takes polynomial time of the number of nodes [4]. However, if graph G has many nodes, this approach is far from practical. For an n-hypercube, an n-star graph and an n-rotator graph, the algorithms of polynomial time of n for this problem have already been proposed [2], [5], [10].

An n-transposition graph [8] is a Cayley graph [1]. It is an $n(n-1)/2$-connected undirected graph with $n!$ nodes and $n(n-1)n!/4$ edges. Its diameter is $n-1$. As an interconnection network, this graph attracts some attention because it can include other topologies as its subgraphs, such as meshes, hypercubes, star graphs and bubble-sort graphs. In addition, the fault diameter of an n-transposition graph is n, and the graph has a wide container between any pair of nodes with length of the distance of them plus two at most.

In this paper, we take an n-transposition graph as a target and propose an algorithm that solves the node-to-set disjoint paths problem in the time complexity of polynomial order of n instead of the number of nodes, $n!$.

The rest of this paper is organized as follows. Section 2 introduces some preliminary definitions and a simple routing algorithm. Section 3 explains our algorithm to obtain node-to-set disjoint paths in detail. In Sect. 4, we give a proof of validity of our algorithm and estimations of its complexities. We conduct computer experiments in Sect. 5. Section 6 describes the conclusion.

2. Preliminaries

In this section, we introduce definitions of the transposition operation, transposition graphs, and the shortest-path routing algorithm in a transposition graph.

Definition 1: For an arbitrary permutation $\boldsymbol{u} = u_1 u_2 \cdots u_n$ of n symbols $1, 2, \cdots, n$, the transposition operation $t_{(i,j)}(\boldsymbol{u})$ ($1 \le i < j \le n$) is defined as follows:

$$t_{(i,j)}(\boldsymbol{u}) = u_1 \cdots u_{i-1} u_j u_{i+1} \cdots u_{j-1} u_i u_{j+1} \cdots u_n.$$

Definition 2: An n-transposition graph, T_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 $\boldsymbol{u} = u_1 u_2 \cdots u_n$ is adjacent to $n(n-1)/2$ nodes whose addresses are elements of the set $\{t_{(i,j)}(\boldsymbol{u}) \mid 1 \le i < j \le n\}$.

In an n-transposition graph T_n, a subgraph induced by nodes that have a common symbol k at the ith position of their addresses constitutes an $(n-1)$-transposition graph. In this paper, we denote the subgraph induced by nodes whose last symbols are k as $T_{n-1}k$. Figure 1 shows some examples of transposition graphs.

For given nodes $\boldsymbol{s} = s_1 s_2 \cdots s_n$ and $\boldsymbol{d} = d_1 d_2 \cdots d_n$ in T_n, we use the routing algorithm `route` shown in Fig. 2 to obtain one of the shortest paths between $\boldsymbol{s}$ and $\boldsymbol{d}$. We assume that the address of a node is represented by using a linear array and each element of the array consists of a word that can store the value n. Then its time complexity is $O(n^2)$ and its path length is $O(n)$.

For an arbitrary node $\boldsymbol{u}$, let $N(\boldsymbol{u})$ denote the set of neighbor nodes of $\boldsymbol{u}$.

3. The Algorithm

In this section, we propose an algorithm for the node-to-set disjoint paths problem in an n-transposition graph.

Manuscript received December 22, 2005.
Manuscript revised June 9, 2006.

[†]The author is with the Fujitsu Access Limited, Kawasaki-shi, 213–8586 Japan.
[††]The authors are 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/e89–d.10.2600

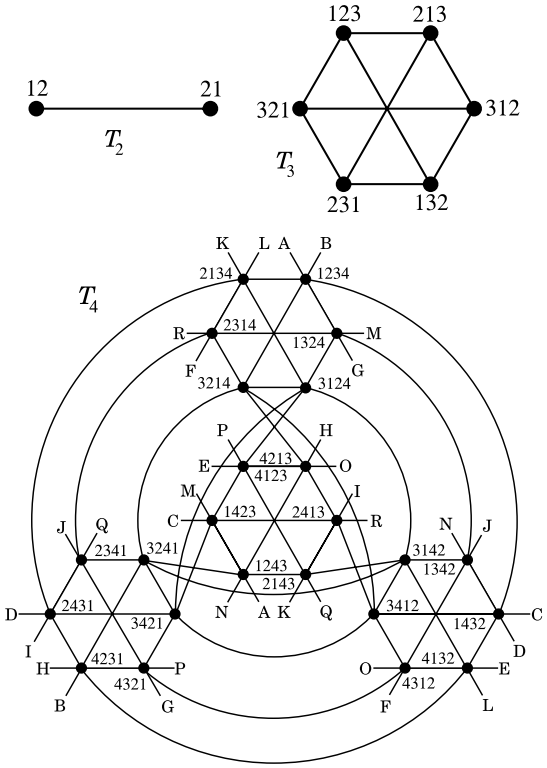

Fig. 1 Examples of transposition graphs.

```
procedure route(s, d);
begin
  c := s; P := [c];
  for i := 1 to n − 1 do
    if c_i <> d_i then begin
      find j such that c_j = d_i;
      c := t_(i,j)(c);
      P := P ++ [c]
    end
end;
```

Fig. 2 A shortest-path routing algorithm `route`.

3.1 Classification

If $n \leq 2$, the problem is trivial. That is, a 2-transposition 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. Therefore, we assume $n \geq 3$ in the following. We can fix the source node as $\boldsymbol{s} = 12\cdots n$, taking advantage of the symmetric property of T_n. Let $D = \{\boldsymbol{d}_1, \boldsymbol{d}_2, \cdots, \boldsymbol{d}_{n(n-1)/2}\}$ be the set of destination nodes. The algorithm has recursive structure. Here let us consider the following two cases.

Case 1 $|D \setminus V(T_{n-1}n)| \leq n-1$
Case 2 $|D \setminus V(T_{n-1}n)| \geq n$

where $V(G)$ represents the node set of G, and $|D \setminus V(T_{n-1}n)|$ represents the number of destination nodes that are not included in $T_{n-1}n$.

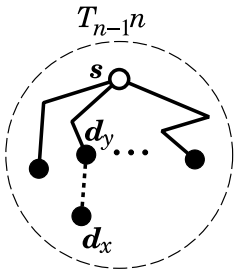

Fig. 3 Construction of node-disjoint paths from $\boldsymbol{s}$ to $(n-1)(n-2)/2$ nodes in $T_{n-1}n$.

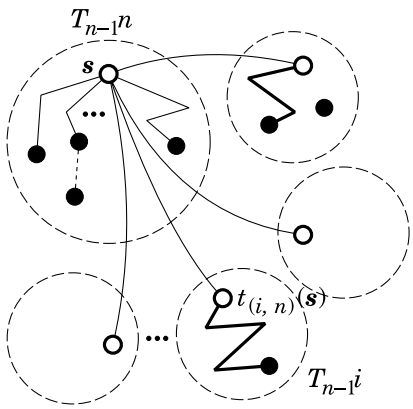

Fig. 4 Construction of paths to the nearest destinations.

3.2 Case 1: $|D \setminus V(T_{n-1}n)| \leq n-1$

This subsection presents the procedure in the case that $|D \setminus V(T_{n-1}n)| \leq n-1$. Note that the number of destination nodes that are included in $T_{n-1}n$ is at least $(n-1)(n-2)/2$ in this case.

Step 1 In $T_{n-1}n$, by calling the algorithm recursively, construct node-disjoint paths from $\boldsymbol{s}$ to $(n-1)(n-2)/2$ arbitrary destination nodes in $T_{n-1}n$.

Step 2 If a destination node, say $\boldsymbol{d}_x$, other than these $(n-1)(n-2)/2$ destination nodes is on one of the constructed path from $\boldsymbol{s}$ to, say $\boldsymbol{d}_y$, then discard the subpath from $\boldsymbol{d}_x$ to $\boldsymbol{d}_y$ and exchange the indices x and y. Repeat this step until no destination node is on the paths except for the $(n-1)(n-2)/2$ nodes. See Fig. 3.

Step 3 Select the edges $(\boldsymbol{s}, t_{(i,n)}(\boldsymbol{s}))$ $(1 \leq i \leq n-1)$. Note that $t_{(i,n)}(\boldsymbol{s}) \in V(T_{n-1}i)$.

Step 4 For each $T_{n-1}i$ $(1 \leq i \leq n-1)$, if there exist some destination nodes in $T_{n-1}i$, choose one of the nearest nodes among them from $t_{(i,n)}(\boldsymbol{s})$. Construct the shortest path between these two nodes by `route` in Fig. 2. See Fig. 4.

Step 5 For each $T_{n-1}i$ $(1 \leq i \leq n-1)$, if there exists no destination node, choose one of the destination nodes to which the path is not yet constructed from $\boldsymbol{s}$. Let the chosen node be $\boldsymbol{d}_z$. Select the edge $(N(\boldsymbol{d}_z) \cap V(T_{n-1}i), \boldsymbol{d}_z)$ and construct the shortest path

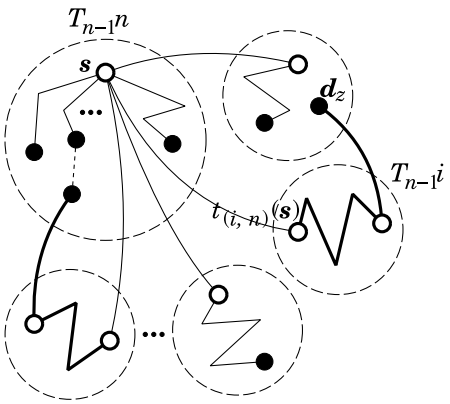

Fig. 5 Construction of paths to destinations to which any path is not yet constructed from s.

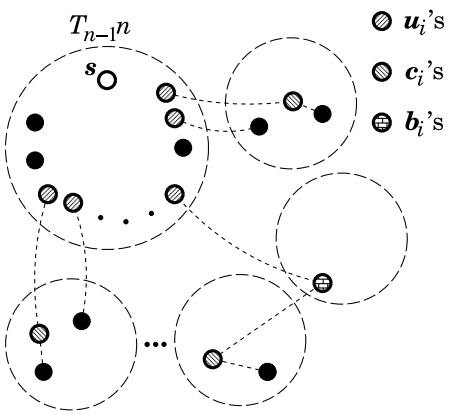

Fig. 6 Selection of u_i, c_i and b_i for each destination d_i.

from $t_{(i,n)}(s)$ to $N(d_z) \cap V(T_{n-1}i)$ by `route` in Fig. 2. See Fig. 5.

3.3 Case 2: $|D \setminus V(T_{n-1}n)| \geq n$

This subsection presents the procedure in the case that $|D \setminus V(T_{n-1}n)| \geq n$.

Step 1 For each destination node d_i outside $T_{n-1}n$, select two nodes u_i and c_i satisfying the following conditions if possible.

- $c_i = d_i$,
- $u_i = (N(c_i) \cap V(T_{n-1}n)) \setminus D$,
- $u_i = s$ or $u_i \neq u_j$ if $i \neq j$.

Step 2 For each destination node d_i outside $T_{n-1}n$, if c_i for d_i was not selected in Step 1, select two nodes u_i and c_i satisfying the following conditions if possible.

- $c_i \in N(d_i) \setminus D$,
- $u_i = (N(c_i) \cap V(T_{n-1}n)) \setminus D$,
- $u_i = s$ or $u_i \neq u_j$ if $i \neq j$,
- $c_i \neq c_j$ if $i \neq j$.

Step 3 For each destination node d_i outside $T_{n-1}n$, if c_i for d_i was not selected in previous steps, select three nodes u_i, c_i and b_i satisfying the following conditions. Figure 6 shows selection of these nodes.

- $c_i \in N(d_i) \setminus D$,
- $b_i \in (N(c_i) \setminus V(T_{n-1}n)) \setminus D$,
- $u_i = (N(b_i) \cap V(T_{n-1}n)) \setminus D$,
- $u_i = s$ or $u_i \neq u_j$ if $i \neq j$,
- $b_i \neq b_j$ if $i \neq j$,
- $c_i \neq c_j$ if $i \neq j$,
- $b_i \neq c_j$ for any i and j.

Step 4 Let M and U be a set $\{d_i \mid d_i \notin V(T_{n-1}n)\} \cup \{c_i \mid c_i \neq d_i\} \cup \{b_i\}$ and a set $\{u_i\}$, respectively.

Step 5 Select the edges $(s, t_{(i,n)}(s))$ $(1 \leq i \leq n-1)$. Note that $t_{(i,n)}(s) \in V(T_{n-1}i)$.

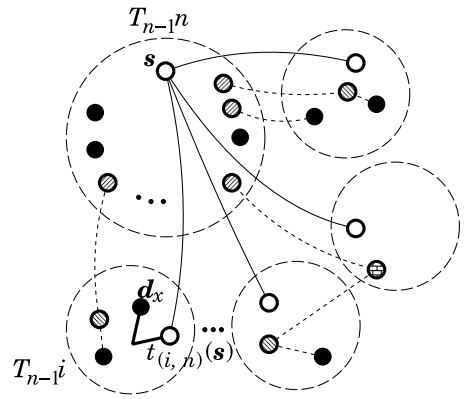

Fig. 7 Construction of paths to the nearest destinations.

Step 6 For each $T_{n-1}i$ $(1 \leq i \leq n-1)$, if there exist some nodes in $M \cap V(T_{n-1}i)$ and a path from $t_{(i,n)}(s)$ is not yet constructed, choose one node v_i among the nodes in $M \cap V(T_{n-1}i)$ such that v_i is one of the nearest nodes from $t_{(i,n)}(s)$ in $M \cap V(T_{n-1}i)$.

Step 7 For each v_i $(1 \leq i \leq n-1)$, if v_i is a destination, say, d_x, construct the shortest path from $t_{(i,n)}(s)$ to d_x by `route` in Fig. 2, and update M and U by $M \setminus \{b_x, c_x, d_x\}$ and $U \setminus \{u_x\}$, respectively. See Fig. 7. In this step, if M is updated, go back to Step 6.

Step 8 For each v_i $(1 \leq i \leq n-1)$, if v_i is one of c_i's, say c_x, construct the shortest path from $t_{(i,n)}(s)$ to c_x by `route` in Fig. 2 and select the edge (c_x, d_x) as shown in Fig. 8, and update M and U by $M \setminus \{b_x, c_x, d_x\}$ and $U \setminus \{u_x\}$, respectively. In this step, if M is updated, go back to Step 6.

Step 9 For each v_i $(1 \leq i \leq n-1)$, v_i is one of b_i's, say b_x. Construct the shortest path from $t_{(i,n)}(s)$ to b_x by `route` in Fig. 2. Update M and U by $M \setminus \{b_x, c_x, d_x\}$ and $U \setminus \{u_x\}$, respectively.

Step 10 For each $T_{n-1}i$ $(1 \leq i \leq n-1)$, if there exists no node in $M \cap V(T_{n-1}i)$ and a path from $t_{(i,n)}(s)$ is not constructed, choose one destination node from M, say d_x, select the edge $(N(d_x) \cap V(T_{n-1}i), d_x)$, construct the shortest path from $t_{(i,n)}(s)$ to $N(d_x) \cap V(T_{n-1}i)$ by `route`

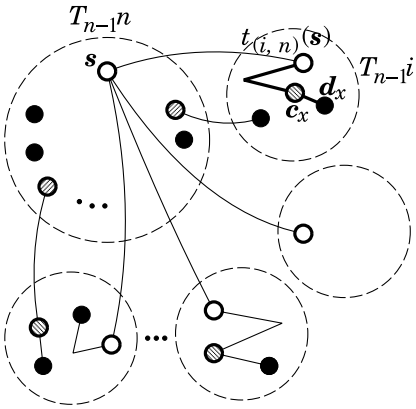

Fig. 8 Construction of paths to the nearest $\boldsymbol{c}_i$'s.

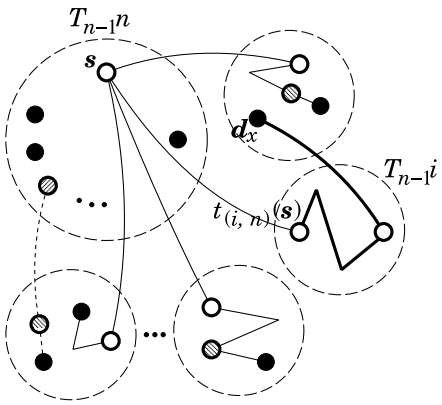

Fig. 9 Construction of paths to destinations outside $T_{n-1}n$ to which any path is not yet constructed from $\boldsymbol{s}$.

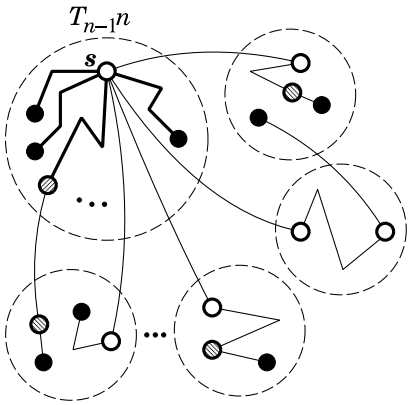

Fig. 10 Recursive application of the algorithm.

in Fig. 2 as shown in Fig. 9, and update M and U by $M \setminus \{\boldsymbol{b}_x, \boldsymbol{c}_x, \boldsymbol{d}_x\}$ and $U \setminus \{\boldsymbol{u}_x\}$.

Step 11 In $T_{n-1}n$, by calling the algorithm recursively, construct node-disjoint paths from $\boldsymbol{s}$ to the nodes in $\{\boldsymbol{d}_i \mid \boldsymbol{d}_i \in T_{n-1}n\} \cup U$ as shown in Fig. 10.

Step 12 For each $\boldsymbol{u}_i$ in U, construct a path from $\boldsymbol{u}_i$ to $\boldsymbol{d}_i$ via $\boldsymbol{b}_i$ and $\boldsymbol{c}_i$ if any.

4. Proof of Validity and Estimation of Complexities

In this section, we give a proof of validity of our algorithm and estimate the time complexity $T(n)$ and the maximum length of each path $L(n)$ for an n-transposition graph. The proof is based on the mathematical induction on n.

Lemma 1: Paths constructed by the procedure for Case 1 are node-disjoint. For this case, the maximum length of each path is $\max\{L(n-1), n\}$ and the time complexity of the procedure is $T(n-1) + L(n-1) \times O(n^4)$.

Proof: In Steps 1 and 2, the obtained $(n-1)(n-2)/2$ paths are node-disjoint except for $\boldsymbol{s}$ by the induction hypothesis. The pair of the edges $(\boldsymbol{s}, t_{(i,n)}(\boldsymbol{s}))$ and $(\boldsymbol{s}, t_{(j,n)}(\boldsymbol{s}))$ selected in Step 3 are node-disjoint except for $\boldsymbol{s}$ if $i \neq j$. The paths in $T_{n-1}i$ and $T_{n-1}j$ constructed in Step 4 are node-disjoint because the selected destination node in each subgraph is the one of the nearest nodes from $t_{(i,n)}(\boldsymbol{s})$ or $t_{(j,n)}(\boldsymbol{s})$. The paths constructed in Step 5 are also node-disjoint. Concatenations of all paths constructed in Steps 3 and 4 or Steps 3 and 5 are node-disjoint except for $\boldsymbol{s}$. The $(n-1)(n-2)/2$ paths obtained in Steps 1 and 2 are all inside $T_{n-1}n$ and $n-1$ paths obtained in Steps 3, 4 and 5 are all outside $T_{n-1}n$ except for $\boldsymbol{s}$ if $|D \setminus V(T_{n-1}n)| = n-1$, or except for $\boldsymbol{s}$ and some destinations otherwise. Hence, the $n(n-1)/2$ paths obtained by the procedure are node-disjoint except for $\boldsymbol{s}$.

The maximum lengths of Steps 1, 3, 4 and 5 are $L(n-1)$, 1, $n-2$ and $n-1$, respectively. Hence, for Case 1, we obtain $L(n) = \max\{L(n-1), n\}$.

The time complexities of Steps 1 and 2 are of $T(n-1)$ and $L(n-1) \times O(n^4)$, respectively. Considering that distance between two nodes in T_n can be calculated in $O(n)$ time, the time complexities of Step 3, 4 and 5 are $O(n)$, $O(n^3)$, $O(n^3)$, respectively. From above, we obtain $T(n) = T(n-1) + L(n-1) \times O(n^4)$ for Case 1. □

Lemma 2: Paths constructed by the procedure for Case 2 are node-disjoint. The maximum length of each path is $\max\{L(n-1)+3, n+1\}$ and the time complexity of the procedure for Case 2 is $T(n-1) + O(n^6)$.

Proof: In T_n, consider the destination node $\boldsymbol{d}_i \notin T_{n-1}n$. Let $N_0(\boldsymbol{d}_i)$, $N_1(\boldsymbol{d}_i)$, and $N_2(\boldsymbol{d}_i)$ represent the sets of the nodes whose distances from $\boldsymbol{d}_i$ are 0, 1, and 2, respectively. Then, $|N_0(\boldsymbol{d}_i)| = 1$, $|N_1(\boldsymbol{d}_i)| = n(n-1)/2$, and $|N_2(\boldsymbol{d}_i)| = n(n-1)(n-2)(3n-1)/24$. In addition, let $\tilde{N}_0(\boldsymbol{d}_i)$ $(\subset N_0(\boldsymbol{d}_i))$, $\tilde{N}_1(\boldsymbol{d}_i)$ $(\subset N_1(\boldsymbol{d}_i))$, and $\tilde{N}_2(\boldsymbol{d}_i)$ $(\subset N_2(\boldsymbol{d}_i))$ represent the sets of the nodes of which the positions of n are same as $\boldsymbol{d}_i$. Then, $|\tilde{N}_0(\boldsymbol{d}_i)| = 1$, $|\tilde{N}_1(\boldsymbol{d}_i)| = (n-1)(n-2)/2$, and $|\tilde{N}_2(\boldsymbol{d}_i)| = (n-1)(n-2)(n-3)(3n-4)/24$.

For any two distinct nodes $\boldsymbol{a}$, $\boldsymbol{b}$ $(\in \tilde{N}_0(\boldsymbol{d}_i) \cup \tilde{N}_1(\boldsymbol{d}_i) \cup \tilde{N}_2(\boldsymbol{d}_i))$, since the positions of n of $\boldsymbol{a}$ and $\boldsymbol{b}$ are same, $N(\boldsymbol{a}) \cap V(T_{n-1}n)$ and $N(\boldsymbol{b}) \cap V(T_{n-1}n)$ are also distinct. Hence, after Steps 1 and 2, at least $(n-1)(n-2)/2 + 1$ destination nodes $\boldsymbol{d}_j$'s have their corresponding $\boldsymbol{u}_j$'s, and the remaining destination nodes are at most $n-2$.

Consider the case where a destination node $\boldsymbol{d}_i =$

$(d_1, d_2, \ldots, d_n)$ is processed in Step 3. Assume that $d_h = n$. Then, for each node in $N_0(\boldsymbol{d}_i) \cup N_1(\boldsymbol{d}_i)$, the node itself is selected as $\boldsymbol{c}_j$, or its neighbor node in $T_{n-1}n$ is selected as $\boldsymbol{u}_j$ for another destination node. If there is an available node in $\tilde{N}_1(\boldsymbol{d}_i)$, select it as $\boldsymbol{c}_i$. Then, $n \geq 4$ implies that the number of candidates for $\boldsymbol{b}_i$, $|N(\boldsymbol{c}_i) \cap N_2(\boldsymbol{d}_i)| = (n-1)(n-2)/2 - 1 \geq n-2$, and that there are paths to distinct $n-2$ nodes in $T_{n-1}n$. If there is not any available node in $\tilde{N}_1(\boldsymbol{d}_i)$, then let $\boldsymbol{c}_i$ be the node that is obtained by exchanging the element $d_h = n$ and the element d_k where $k \neq h, n$. Let $\boldsymbol{b}_i$ be the node obtained by exchanging the element d_k and the element d_l where $l \neq h, k$. There are $n-2$ candidates for $\boldsymbol{b}_i$. Then, the path $\boldsymbol{d}_i \rightarrow \boldsymbol{c}_i \rightarrow \boldsymbol{b}_i$ does not have any common node with $\tilde{N}_0(\boldsymbol{d}_i) \cup \tilde{N}_1(\boldsymbol{d}_i)$, and the node in $N(\boldsymbol{b}_i) \cap V(T_{n-1}n)$ is adjacent to the node in $\tilde{N}_2(\boldsymbol{d}_i)$. Hence, there are paths whose terminal nodes are distinct $n-2$ nodes in $T_{n-1}n$.

The above discussion ensures the existence of a path from $\boldsymbol{d}_i$ to $\boldsymbol{u}_i$ via $\boldsymbol{c}_i$ and $\boldsymbol{b}_i$ for each $\boldsymbol{d}_i$ of the $n-2$ remaining destination nodes in Step 3 of Case 2. That is, the algorithm can always select $\boldsymbol{c}_i$, $\boldsymbol{b}_i$, and $\boldsymbol{u}_i$ for each $\boldsymbol{d}_i$. Note that if there are a node in $\tilde{N}_1(\boldsymbol{d}_i)$ and a node in $\tilde{N}_2(\boldsymbol{d}_i)$ whose corresponding neighbor nodes in $T_{n-1}n$ are not selected as $\boldsymbol{u}_j$'s, they are not used simultaneously for a single destination node.

If i's are different, the edges selected in Step 5 are disjoint except for $\boldsymbol{s}$. Paths constructed in Steps from 7 to 10 are disjoint because of the conditions described in Steps 1, 2 and 3. The $(n-1)(n-2)/2$ paths constructed in Step 11 are node-disjoint by the induction hypothesis. Hence, the $n(n-1)/2$ paths which include a collection of $(n-1)(n-2)/2$ concatenations of paths selected in Step 11, and $n-1$ paths constructed in Steps from 5 to 10, are node-disjoint except for $\boldsymbol{s}$.

The maximum lengths of Step 5 and Steps from 7 to 12 are $1, n-2, n-1, n, n-1, L(n-1)$ and 3, respectively. Hence, for Case 2, we obtain $L(n) = \max\{L(n-1)+3, n+1\}$.

It takes $O(n^6)$ time to select $\boldsymbol{u}_i$'s, $\boldsymbol{c}_i$'s and $\boldsymbol{b}_i$'s for $\boldsymbol{d}_i$'s in Steps 1, 2 and 3. Considering that a distance between two nodes in T_n can be calculated in $O(n)$ time, the time complexities of Steps from 5 to 12 are $O(n)$, $O(n^4)$, $O(n^4)$, $O(n^4)$, $O(n^4)$, $O(n^3)$, $T(n-1)$ and $O(n^2)$ in this order. Steps from 6 to 8 repeat at most $O(n^2)$ times. From discussions above, we obtain $T(n) = T(n-1) + O(n^6)$ for Case 2. □

Lemmas 1 and 2 give the following theorem.

Theorem 1: For an n-transposition graph, $n(n-1)/2$ paths constructed by our algorithm are node-disjoint except for $\boldsymbol{s}$. The time complexity and the maximum length of each path are $O(n^7)$ and $3n-5$, respectively.

5. Computer Experiment

To evaluate the average performance of the algorithm, we conducted the following computer experiment for an n-transposition graph. The algorithm is implemented in the programming language C. The program is compiled by `gcc` with `-O2` option and executed on a target machine with an Intel Celeron 400 MHz CPU and a 128 MB memory unit.

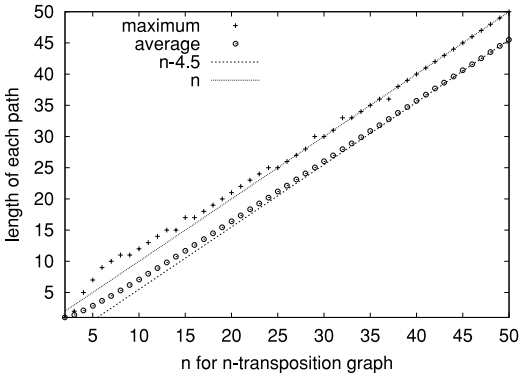

Fig. 11 Length of each path.

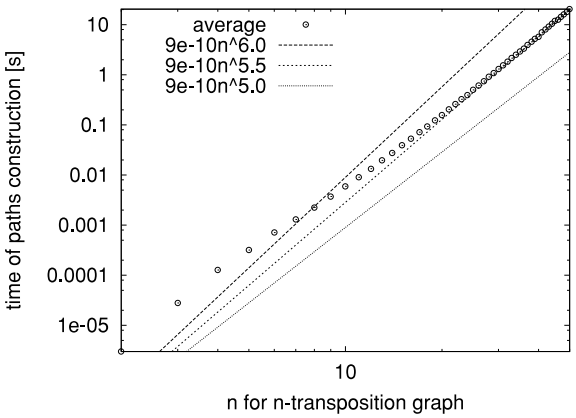

Fig. 12 Time of paths construction.

1. Fix a source node to be $12\cdots n$ and select destination nodes randomly other than the source.
2. Apply the algorithm and measure the length of each path and execution time.

Experiment is performed 1,000 times for each n from 2 to 50. Results are shown in Figs. 11 and 12. From these figures we can observe that the average length of each path and the average time of paths construction are of polynomial order and approximately $O(n)$ and $O(n^{5.5})$, respectively, in their ranges.

6. Conclusions

In this paper, we proposed a polynomial algorithm for the node-to-set disjoint paths problem in an n-transposition graph whose time complexity and the maximum length of each path are $O(n^7)$ and $3n-5$, respectively. We also conducted computer experiments to show that the average length of each path and the average time are $O(n)$ and $O(n^{5.5})$, respectively.

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] 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.–Oct. 1992.
[3] M. Dietzfelbinger, S. Madhavapeddy, and I.H. Sudborough, "Three disjoint path paradigms in star networks," Proc. Third IEEE Symposium on Parallel and Distributed Processing, pp.400–406, 1991.
[4] J.T. Gross and J. Yellen, Graph theory and its applications, CRC Press, 1998.
[5] Q. 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.
[6] 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.
[7] 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.
[8] S. Latifi and P.K. Srimani, "Transposition networks as a class of fault-tolerant robust networks," IEEE Trans. Comput., vol.45, no.2, pp.230–238, July 1996.
[9] S. Madhavapeddy and I.H. Sudborough, "A topological property of hypercubes: Node disjoint paths," Proc. Second IEEE Symposium on Parallel and Distributed Processing, pp.532–539, 1990.
[10] M.O. Rabin, "Efficient dispersal of information for security, load balancing, and fault tolerance," J. Association for Computing Machinery, vol.36, no.2, pp.335–348, April 1989.
[11] P.D. Seymour, "Disjoint paths in graphs," Discrete Math., vol.29, pp.293–309, 1980.

Yasuto Suzuki received the B.E., M.E. and Ph.D. degrees from Tokyo University of Agriculture and Technology in 2001, 2003 and 2006, respectively. He joined Fujitsu Access Limited in 2006. His research interests include graph and network theories and fault-tolerant systems.

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 a member of ACM, IPSJ and JSSST.

Mario Nakamori is a Professor at Tokyo University of Agriculture and Technology. His main research areas are algorithmics, mathematical programming, and their applications. He received the B.E., M.E. and D.E. degrees from the University of Tokyo in 1971, 1973 and 1976, respectively. He is a member of ACM, IPSJ, and ORSJ.