社団法人 情報処理学会　研究報告
IPSJ SIG Technical Report

2004－MPS－50　(4)
2004／6／22

トランスポジショングラフにおける素な経路

鈴木 康斗, 金子 敬一, 中森 眞理雄
東京農工大学

本論文では, n-トランスポジショングラフにおいて頂点から頂点集合への互いに素な経路問題に対する n の多項式時間のアルゴリズムを提案する. アルゴリズムは再帰的に記述され, 目的頂点の位置によりふたつの場合に分けられる. アルゴリズムが与える経路の長さの最大値と時間計算量のオーダを見積もって示す. また, 計算機実験により提案アルゴリズムの性能評価を行う.

Node-disjoint Paths in a Transposition Graph

Yasuto Suzuki, Keiichi Kaneko and Mario Nakamori
Tokyo University of Agriculture and Technology

In this paper, we give an algorithm for the node-to-set disjoint paths problem in transposition graphs. 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 and the average performance is evaluated based on computer experiment.

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, 2, 5] have been proposed to replace simple networks such as hypercubes and meshes. A transposition graph[5] provides one such new topology. It can include other topologies as its subgraphs, such as hypercubes, star graphs and bubble-sort graphs.

Unfortunately, there still remain unknowns in several metrics for this topology despite intensive research activities. 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, find k paths from $\boldsymbol{s}$ to each $\boldsymbol{d}_i$ that 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[3, 4, 6]. Once these k paths are obtained, they achieve some 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 the number of the nodes by the maximum flow algorithm. However, in an n-transposition 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 $n!$.

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 $\boldsymbol{u} = u_1u_2\cdots u_n$ 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:

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

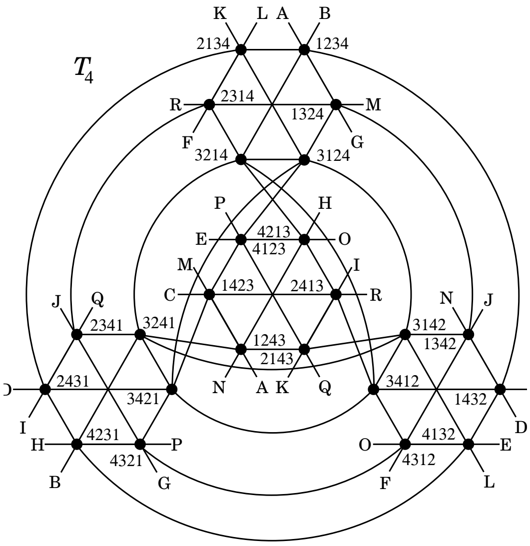

Figure 1: An example of transposition graph.

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 $\{t_{(i,j)}(\boldsymbol{u}) \mid 1 \leq i < j \leq n\}$.

Figure 1 shows an example of transposition graph. 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 $T_{n-1}k$. 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 Figure 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 T_n.

```
procedure route(s, d);
begin
  c := s; P := [c];
  for i := 1 to n−1
    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
end;
```

Figure 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 and it is divided into two procedures depending on $|D \backslash T_{n-1}n|$ where $|D \backslash T_{n-1}n|$ represents the number of destination nodes that are not included in $T_{n-1}n$.

3.2 Case 1: $|D \backslash T_{n-1}n| \leq n-1$

This subsection presents the procedure for the case that $|D \backslash T_{n-1}n| \leq n-1$.

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.

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

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.

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 an edge $(N(\boldsymbol{d}_z) \cap T_{n-1}i, \boldsymbol{d}_z)$ and construct the shortest path from $t_{(i,n)}(\boldsymbol{s})$ to $N(\boldsymbol{d}_z) \cap T_{n-1}i$.

3.3 Case 2: $|D \backslash T_{n-1}n| \geq n$

This subsection presents the procedure for the case that $|D \backslash T_{n-1}n| \geq n$.

Step 1 For each destination node $\boldsymbol{d}_i$ outside $T_{n-1}n$, select two nodes $\boldsymbol{u}_i$ and $\boldsymbol{c}_i$ satisfying the following conditions if possible.

- $\boldsymbol{c}_i = \boldsymbol{d}_i$,
- $\boldsymbol{u}_i = (N(\boldsymbol{c}_i) \cap T_{n-1}n) \backslash D$,
- $\boldsymbol{u}_i = \boldsymbol{s}$ or $\boldsymbol{u}_i \neq \boldsymbol{u}_j$ if $i \neq j$.

Step 2 For each destination node $\boldsymbol{d}_i$ outside $T_{n-1}n$, if $\boldsymbol{c}_i$ for $\boldsymbol{d}_i$ was not selected in Step 1, select two nodes $\boldsymbol{u}_i$ and $\boldsymbol{c}_i$ satisfying the following conditions if possible.

- $\boldsymbol{c}_i \in N(\boldsymbol{d}_i) \backslash D$,
- $\boldsymbol{u}_i = (N(\boldsymbol{c}_i) \cap T_{n-1}n) \backslash D$,
- $\boldsymbol{u}_i = \boldsymbol{s}$ or $\boldsymbol{u}_i \neq \boldsymbol{u}_j$ if $i \neq j$,
- $\boldsymbol{c}_i \neq \boldsymbol{c}_j$ if $i \neq j$.

Step 3 For each destination node $\boldsymbol{d}_i$ outside $T_{n-1}n$, if $\boldsymbol{c}_i$ for $\boldsymbol{d}_i$ was not selected in previous steps, select three nodes $\boldsymbol{u}_i$, $\boldsymbol{c}_i$ and $\boldsymbol{b}_i$ satisfying the following conditions.

- $\boldsymbol{c}_i \in N(\boldsymbol{d}_i) \backslash D$,
- $\boldsymbol{b}_i \in (N(\boldsymbol{c}_i) \backslash T_{n-1}n) \backslash D$,
- $\boldsymbol{u}_i = (N(\boldsymbol{b}_i) \cap T_{n-1}n) \backslash D$,
- $\boldsymbol{u}_i = \boldsymbol{s}$ or $\boldsymbol{u}_i \neq \boldsymbol{u}_j$ if $i \neq j$,
- $\boldsymbol{b}_i \neq \boldsymbol{b}_j$ if $i \neq j$,
- $\boldsymbol{c}_i \neq \boldsymbol{c}_j$ if $i \neq j$,
- $\boldsymbol{b}_i \neq \boldsymbol{c}_j$ for any i and j.

Step 4 Let M and U be a set $\{\boldsymbol{d}_i \mid \boldsymbol{d}_i \notin T_{n-1}n\} \cup \{\boldsymbol{c}_i \mid \boldsymbol{c}_i \neq \boldsymbol{d}_i\} \cup \{\boldsymbol{b}_i\}$ and a set $\{\boldsymbol{u}_i\}$, respectively.

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

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

Step 7 For each $\boldsymbol{v}_i$ $(1 \leq i \leq n-1)$, if $\boldsymbol{v}_i$ is a destination, say, $\boldsymbol{d}_x$, construct the shortest path from $t_{(i,n)}(\boldsymbol{s})$ to $\boldsymbol{d}_x$, and update M and U by $M \backslash \{\boldsymbol{b}_x, \boldsymbol{c}_x, \boldsymbol{d}_x\}$ and $U \backslash \{\boldsymbol{u}_x\}$, respectively. In this step, if M is updated, go back to Step 6.

Step 8 For each $\boldsymbol{v}_i$ $(1 \leq i \leq n-1)$, if $\boldsymbol{v}_i$ is one of $\boldsymbol{c}_i$'s, say, $\boldsymbol{c}_x$, construct the shortest path from $t_{(i,n)}(\boldsymbol{s})$ to $\boldsymbol{c}_x$ and select an edge $(\boldsymbol{c}_x, \boldsymbol{d}_x)$, and update M and U by $M \backslash \{\boldsymbol{b}_x, \boldsymbol{c}_x, \boldsymbol{d}_x\}$ and $U \backslash \{\boldsymbol{u}_x\}$, respectively. In this step, if M is updated, go back to Step 6.

Step 9 For each $\boldsymbol{v}_i$ $(1 \leq i \leq n-1)$, $\boldsymbol{v}_i$ is one of $\boldsymbol{b}_i$'s, say, $\boldsymbol{b}_x$. Construct the shortest path from $t_{(i,n)}(\boldsymbol{s})$ to $\boldsymbol{b}_x$. Update M and U by $M \backslash \{\boldsymbol{b}_x, \boldsymbol{c}_x, \boldsymbol{d}_x\}$ and $U \backslash \{\boldsymbol{u}_x\}$, respectively.

Step 10 For each $T_{n-1}i$ $(1 \leq i \leq n-1)$, if there exists no node in $M \cap T_{n-1}i$ and a path from $t_{(i,n)}(\boldsymbol{s})$ is not constructed, choose one destination node from M, say, $\boldsymbol{d}_x$, select an edge $(N(\boldsymbol{d}_x) \cap T_{n-1}i, \boldsymbol{d}_x)$, construct the shortest path from $t_{(i,n)}(\boldsymbol{s})$ to $N(\boldsymbol{d}_x) \cap T_{n-1}i$, and update M and U by $M \backslash \{\boldsymbol{b}_x, \boldsymbol{c}_x, \boldsymbol{d}_x\}$ and $U \backslash \{\boldsymbol{u}_x\}$.

Step 11 In $T_{n-1}n$, by calling the algorithm recursively, construct node-disjoint paths from $\boldsymbol{s}$ to the nodes in $(D \cap T_{n-1}n) \cup U$.

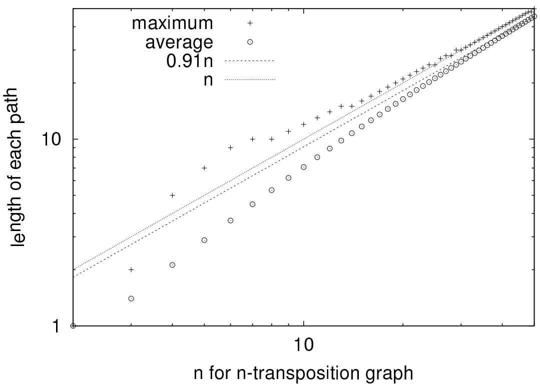

Figure 3: Length of each path.

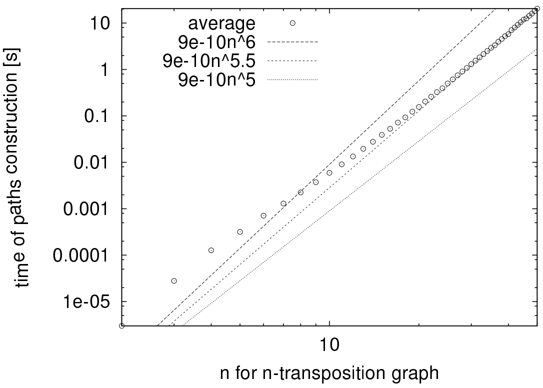

Figure 4: Time of paths construction.

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.

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.

4 Computer experiment

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

1. Fix the source to be $12\cdots n$ and select destinations 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 Figure 3 and Figure 4. 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 $O(n^{5.5})$ in their ranges.

5 Conclusions

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

Acknowledgement

This work was partly supported by Grant-in-Aid for Scientific Research (C) of JSPS under Grant No. 16500015 and Grant-in-Aid for JSPS Fellows.

References

[1] S.B. Akers et *al.*, "A group-theoretic model for symmetric interconnection networks," *IEEE Trans. Comp.*, 38(4):555–566, 1989.

[2] P.F. Corbett, "Rotator graphs: An efficient topology for point-to-point multiprocessor networks," *IEEE Trans. Parallel & Distributed Syst.*, 3(5):622–626, 1992.

[3] Q. Gu et *al.*, "Node-to-set disjoint paths problem in star graphs," *Inf. Proc. Lett.*, 62(4):201–207, 1997.

[4] K. Kaneko et *al.*, "Node-to-set disjoint paths problem in pancake graphs," *IEICE Trans. Inf. & Syst.*, E86-D(9):1628–1633, 2003.

[5] S. Latifi et al., "Transposition networks as a class of fault-tolerant robust networks," *IEEE Trans. Comp.*, 45(3):230–238, 1996.

[6] M.O. Rabin, "Efficient dispersal of information for security, load balancing, and fault tolerance," *JACM*, 36(2):335–348, 1989.