IEICE TRANS. INF. & SYST., VOL.E91–D, NO.4 APRIL 2008

PAPER

The Container Problem in Bubble-Sort Graphs

Yasuto SUZUKI†, *Nonmember and* Keiichi KANEKO†a), *Member*

SUMMARY Bubble-sort graphs are variants of Cayley graphs. A bubble-sort graph is suitable as a topology for massively parallel systems because of its simple and regular structure. Therefore, in this study, we focus on n-bubble-sort graphs and propose an algorithm to obtain $n-1$ disjoint paths between two arbitrary nodes in time bounded by a polynomial in n, the degree of the graph plus one. We estimate the time complexity of the algorithm and the sum of the path lengths after proving the correctness of the algorithm. In addition, we report the results of computer experiments evaluating the average performance of the algorithm.

key words: *bubble-sort graphs, internally disjoint paths, polynomial algorithm, fault tolerance*

1. Introduction

Recently, studies on parallel computation are becoming increasingly important, and many topologies have been proposed to interconnect computers. A bubble-sort graph [1] is one such topology and it is a variant of a notion found in Cayley graphs. The bubble-sort graph is studied because of its simple and regular structure [2], [20].

The container problem is to find k paths between two arbitrary nodes $\boldsymbol{a}$ and $\boldsymbol{b}$ in a k-connected graph such that those paths are internally disjoint, that is, they do not have common nodes except for nodes $\boldsymbol{a}$ and $\boldsymbol{b}$. This is an important issue in parallel computation as well as the node-to-set disjoint paths problem [7], [9], [10], [16]. In this study, we select the bubble-sort graph as the target and present a solution to the container problem. In certain Cayley graphs, in particular for a hypercube [17], a rotator graph [4], a pancake graph [1], and a star graph [1], polynomial time algorithms have been proposed to obtain internally disjoint paths [5], [8], [15], [18]. Suzuki and Kaneko [19] have solved the node-to-set disjoint paths problem in bubble-srt graphs in $O(n^5)$. Additionally, Latifi and Srimani [14] have solved the container problem in the extended graph, which is obtained by adding redundant links to a bubble-sort graph to augment the fault tolerance.

The rest of this paper is structured as follows. In Sect. 2, the definitions of bubble-sort graphs and the shortest-path routing algorithm for them are introduced. We explain our algorithm in detail in Sect. 3. Section 4 describes a proof of correctness and provides an estimate of the complexity of our algorithm. In Sect. 5, experiments for performance evaluation and associated results are presented. Finally, conclusions are given in Sect. 6.

Manuscript received May 30, 2007.
Manuscript revised October 12, 2007.

†The authors are with the Graduate School of Engineering, Tokyo University of Agriculture and Technology, Koganei-shi, 184–8588 Japan.
a) E-mail: klkaneko@cc.tuat.ac.jp

DOI: 10.1093/ietisy/e91–d.4.1003

2. Preliminaries

Definition 1: (Exchange Operation) For a permutation $\boldsymbol{a} = (a_1, a_2, \ldots, a_n)$ of $1, 2, \ldots, n$, we define the exchange operation by

$$\boldsymbol{a}^{(i)} = (a_1, a_2, \ldots, a_{i-1}, a_{i+1}, a_i, a_{i+2}, \ldots, a_n) \quad (1 \le i \le n-1).$$

Definition 2: (n-bubble-sort graph B_n) B_n is an undirected simple graph that has $n!$ nodes. Each node has a unique label that consists of a permutation $(a_1, a_2, \ldots, a_n)$ of $1, 2, \ldots, n$. Two nodes $\boldsymbol{a}$ and $\boldsymbol{b}$ are adjacent if and only if $\boldsymbol{a}^{(i)} = \boldsymbol{b}$ for some i.

Figure 1 shows the 4-bubble-sort graph B_4 as an example.

The number of nodes, the degree, and the diameter of an n-bubble-sort graph are $n!$, $n-1$, and $n(n-1)/2$, respectively [1], [13]. In the following, if we can construct $n-1$ internally disjoint paths between any two nodes in polynomial time, we will find the solution to the container problem.

Table 1 shows comparisons between B_n and other graphs. In the table, $T_{n,k}$, Q_n, P_n, S_n, R_n, $dB_{n,k}$ and $K_{n,k}$ stand for a k-ary n-dimensional torus, an n-cube, an n-pancake graph, an n-star graph, an n-rotator digraph, an (n,k)-de Bruijn graph [3], and an (n,k)-Kautz graph [3], respectively. A bubble-sort graph is undirected, symmetric, and recursive.

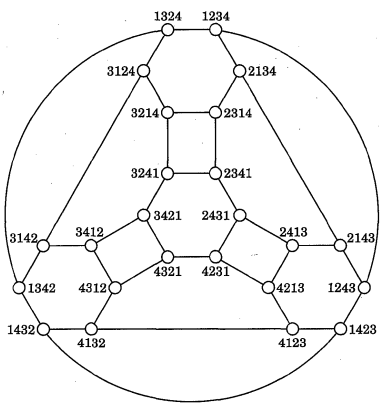


Fig. 1 An example bubble-sort graph B_4.

Table 1 Comparison between bubble-sort graphs and others.

graph	max. deg.	#nodes	diam.	undir.	sym.	rec.
B_n	$n-1$	$n!$	$\frac{n(n-1)}{2}$	Yes	Yes	Yes
$T_{n,k}$	4	k^n	$n \times \lfloor \frac{k}{2} \rfloor$	Yes	Yes	No
Q_n	n	2^n	n	Yes	Yes	Yes
P_n	$n-1$	$n!$	$\leq \lfloor \frac{5(n+1)}{3} \rfloor^{\dagger}$	Yes	Yes	Yes
S_n	$n-1$	$n!$	$\lfloor \frac{3(n-1)}{2} \rfloor$	Yes	Yes	Yes
R_n	$n-1$	$n!$	$n-1$	No	Yes	Yes
$dB_{n,k}$	n	n^k	k	Yes	No	No
$K_{n,k}$	n	$n^k + n^{k-1}$	k	Yes	No	No

†: From [6].

```
shortest(c = (c_1, c_2, ..., c_n), d = (d_1, d_2, ..., d_n))
/* Input: c: current node, d: destination node */
/* Output: P: path from c to d */
begin
  P := [c];
  for i := n to 1 step -1 do
    if c_i <> d_i then begin
      find k such that c_k = d_i;
      for j := k to i - 1 do begin
        c := c^(j);
        P := P ++ [c]
      end
    end
end;
```

Fig. 2 Shortest-path routing algorithm for bubble-sort graphs.

These properties are very important for topologies of interconnection networks used for parallel and distributed computation. The graphs $T_{n,k}$, R_n, $dB_{n,k}$, and $K_{n,k}$ lack at least one of these properties. There is a polynomial time shortest-path routing algorithm for B_n, but it is unknown for P_n. Moreover, B_{2n} has $(2n)!/2^n$ copies of Q_n as subgraphs, while S_n does not.

The n-bubble-sort graph B_n contains n disjoint $(n-1)$-bubble-sort graphs, each of which can be induced by the nodes that are specified by fixing the final integer in their labels. Let us denote each subgraph B_{n-1} by $B_{n-1}h$ by specifying the common integer h.

We show a shortest-path routing algorithm between two nodes in an n-bubble-sort graph in Fig. 2, where the operator ++ performs the concatenation of two lists. This routing algorithm, called `shortest`, is based on the bubble sort. Here we assume that the label of a node is expressed by a linear array of length n. We also assume that the labels of nodes on a path are all memorized in generating the path. Based on these assumptions, the time complexity of `shortest` is $O(n^3)$ and the path length obtained is at most $n(n-1)/2$. Refer to Chapter 8 in [11] or Exercise 36 of Section 5.3.4 in [12] for the correctness and the complexities with respect to `shortest`.

3. Algorithm

In this section, we give an explanation of our algorithm, which solves the container problem in an n-bubble-sort graph in the polynomial order of n.

From the symmetric property of B_n, we fix the source node s to $(1, 2, \ldots, n)$ without loss of generality. Let the destination node be $\boldsymbol{d} = (d_1, d_2, \ldots, d_n)$. If $n = 3$, that is, in B_3, the internally disjoint paths are trivial. Hence, we assume that $n > 3$ in the rest of this paper. The algorithm is divided into the following five cases.

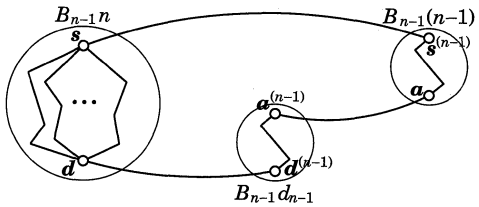


Fig. 3 Case I, Step 6 ($d_{n-1} \neq n-1$).

Case I $d_n = n$.
Case II $d_n \neq n-1$, $d_{n-1} = n$.
Case III $d_n, d_{n-1} \neq n$.
Case IV $s^{(n-1)} = \boldsymbol{d}$.
Case V $d_n = n-1$, $d_{n-1} = n$, $s^{(n-1)} \neq \boldsymbol{d}$.

3.1 Case I

In this case, we assume $d_n = n$.

Step 1 Apply the algorithm recursively in $B_{n-1}n$ to obtain $n-2$ internally disjoint paths from s to $\boldsymbol{d}$.
Step 2 Select edge $(s, s^{(n-1)})$. If $d_{n-1} = n-1$, apply `shortest` in $B_{n-1}(n-1)$ to obtain the path from $s^{(n-1)}$ to $\boldsymbol{d}^{(n-1)}$, and go to Step 6.
Step 3 Apply `shortest` in $B_{n-1}(n-1)$ to obtain the path from $s^{(n-1)}$ to $\boldsymbol{a} = (1, 2, \ldots, d_{n-1}-1, d_{n-1}+1, \ldots, n-2, n, d_{n-1}, n-1)$.
Step 4 Select edge $(\boldsymbol{a}, \boldsymbol{a}^{(n-1)} = (1, 2, \ldots, d_{n-1}-1, d_{n-1}+1, \ldots, n-2, n, n-1, d_{n-1}))$.
Step 5 Apply `shortest` in $B_{n-1}d_{n-1}$ to obtain the path from $\boldsymbol{a}^{(n-1)}$ to $\boldsymbol{d}^{(n-1)} = (d_1, d_2, \ldots, d_{n-2}, d_n, d_{n-1})$.
Step 6 Select edge $(\boldsymbol{d}^{(n-1)}, \boldsymbol{d})$.

Figure 3 shows the internally disjoint paths between s and $\boldsymbol{d}$ after Step 6 in Case I, where $d_{n-1} \neq n-1$.

3.2 Case II

We assume $d_n \neq n-1$ and $d_{n-1} = n$ in this case.

Step 1 For each i $(1 \leq i \leq n-2)$, construct the path from s to $\boldsymbol{c}_i = (1, 2, \ldots, i-1, i+1, \ldots, n-1, i, n)$. Let $\boldsymbol{c}_{n-1} = s$. For $\boldsymbol{c}_{d_n}$, obtain the shortest path to $\boldsymbol{d}^{(n-1)}$ and redefine $\boldsymbol{c}_{d_n} = \boldsymbol{d}^{(n-1)}$.
Step 2 For each i $(1 \leq i \leq n-1)$, select edge $(\boldsymbol{c}_i, \boldsymbol{c}_i^{(n-1)})$. Note that $\boldsymbol{c}_{d_n}^{(n-1)} = \boldsymbol{d}$.
Step 3 For each i $(1 \leq i \leq n-1, i \neq d_n)$, let $\boldsymbol{b}_i = (d_1, d_2, \ldots, d_{l_i-1}, d_{l_i+1}, \ldots, d_{n-1}, i, d_n)$ where $d_{l_i} = i$, and apply `shortest` in $B_{n-1}i$, to obtain the path from $\boldsymbol{c}_i^{(n-1)}$ to $\boldsymbol{b}_i^{(n-1)}$.
Step 4 For each i $(1 \leq i \leq n-1, i \neq d_n)$, select $(\boldsymbol{b}_i^{(n-1)}, \boldsymbol{b}_i)$.

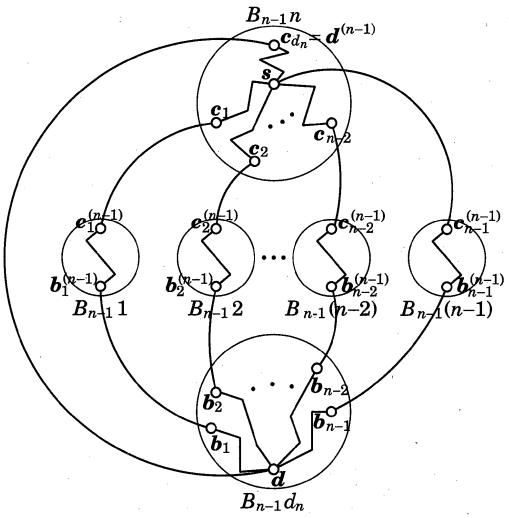


Fig. 4 Case II, Step 5.

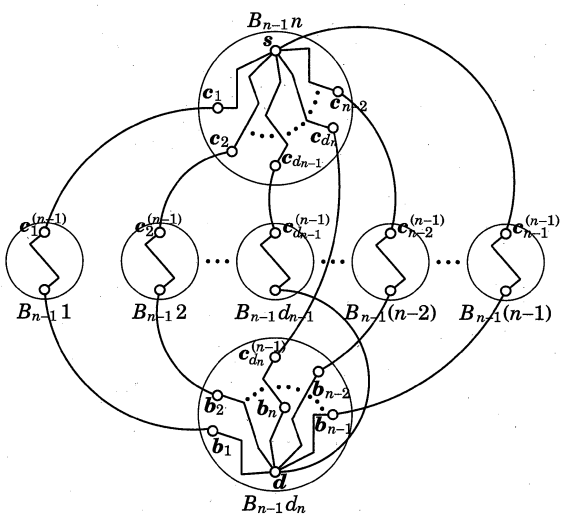


Fig. 5 Case III, Step 6.

Step 5 For each i ($1 \le i \le n-1$, $i \ne d_{n-1}$), select the shortest path from $\boldsymbol{b}_i$ to $\boldsymbol{d}$ in $B_{n-1}d_n$.

Figure 4 shows the internally disjoint paths between $\boldsymbol{s}$ and $\boldsymbol{d}$ after Step 5 in Case II.

3.3 Case III

We assume $d_n, d_{n-1} \ne n$ in this case.

Step 1 For each i ($1 \le i \le n-2$), construct the path from $\boldsymbol{s}$ to $\boldsymbol{c}_i = (1, 2, \ldots, i-1, i+1, , n-1, i, n)$. Let $\boldsymbol{c}_{n-1} = \boldsymbol{s}$.
Step 2 For each i ($1 \le i \le n-1$), select edge $(\boldsymbol{c}_i, \boldsymbol{c}_i^{(n-1)})$.
Step 3 For each i ($1 \le i \le n-1$, $i \ne d_n$), let $\boldsymbol{b}_i = (d_1, d_2, \ldots, d_{l_i-1}, d_{l_i+1}, \ldots, d_{n-1}, i, d_n)$ where $d_{l_i} = i$, and apply `shortest` in $B_{n-1}i$ to obtain the path from $\boldsymbol{c}_i^{(n-1)}$ to $\boldsymbol{b}_i^{(n-1)}$.
Step 4 For each i ($1 \le i \le n-1$, $i \ne d_n$), select edge $(\boldsymbol{b}_i^{(n-1)}, \boldsymbol{b}_i)$. Note that $\boldsymbol{b}_{d_{n-1}} = \boldsymbol{d}$.
Step 5 In $B_{n-1}d_n$, select the shortest path from $\boldsymbol{c}_{d_n}^{(n-1)}$ to $\boldsymbol{b}_{d_n} = (d_1, d_2, \ldots, d_{l_n-1}, d_{l_n+1}, \cdots, d_{n-1}, n, d_n)$ where $d_{l_n} = n$.
Step 6 For each i ($1 \le i \le n-1$, $i \ne d_{n-1}$), select the shortest path from $\boldsymbol{b}_i$ to $\boldsymbol{d}$ in $B_{n-1}d_n$.

Figure 5 shows the internally disjoint paths between $\boldsymbol{s}$ and $\boldsymbol{d}$ after Step 6 in Case III.

3.4 Case IV

In this case, we assume $\boldsymbol{s}^{(n-1)} = \boldsymbol{d}$.

Step 1 Select edge $(\boldsymbol{s}, \boldsymbol{d})$.
Step 2 For each i ($1 \le i \le n-2$), select the shortest path from $\boldsymbol{s}$ to $\boldsymbol{c}_i = (1, 2, \ldots, i-1, i+1, \ldots, n-1, i, n)$.
Step 3 For each i ($1 \le i \le n-2$), select edge $(\boldsymbol{c}_i, \boldsymbol{c}_i^{(n-1)})$.
Step 4 For each i ($1 \le i \le n-2$), let $\boldsymbol{b}_i = (d_1, d_2, \ldots, d_{l_i-1}, d_{l_i+1}, \ldots, d_{n-1}, i, d_n)$ where $d_{l_i} = i$, and apply `shortest` in $B_{n-1}i$ to obtain the path from $\boldsymbol{c}_i^{(n-1)}$ to $\boldsymbol{b}_i^{(n-1)}$.

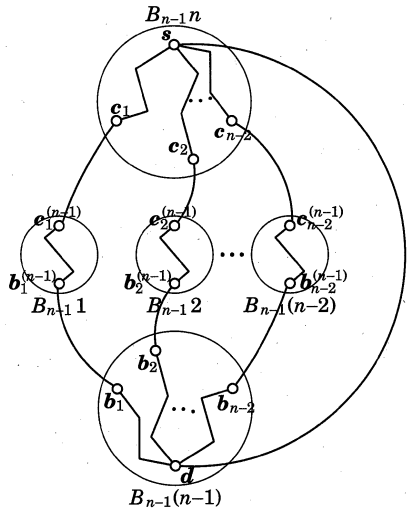


Fig. 6 Case IV, Step 6.

Step 5 For each i ($1 \le i \le n-2$), select edge $(\boldsymbol{b}_i^{(n-1)}, \boldsymbol{b}_i)$.
Step 6 For each i ($1 \le i \le n-2$), select the shortest path from $\boldsymbol{b}_i$ to $\boldsymbol{d}$.

Figure 6 shows the internally disjoint paths between $\boldsymbol{s}$ and $\boldsymbol{d}$ after Step 6 in Case IV.

3.5 Case V

We assume $d_n = n-1$, $d_{n-1} = n$, and $\boldsymbol{s}^{(n-1)} \ne \boldsymbol{d}$ in this case.

Step 1 Find h such that $d_h \ne h$ and $d_i = i$ ($h < i \le n-2$). In the following, we assume $k = d_h$ and $d_j = h$.
Step 2 Apply `shortest` in $B_{n-1}n$ to obtain the path from $\boldsymbol{s}$ to $\boldsymbol{d}^{(n-1)} = (d_1, d_2, \ldots, d_{n-2}, d_n, d_{n-1})$. Select edge $(\boldsymbol{d}^{(n-1)}, \boldsymbol{d})$.
Step 3 Apply `shortest` in $B_{n-1}(n-1)$ to obtain the path from $\boldsymbol{d}$ to $\boldsymbol{s}^{(n-1)} = (1, 2, \ldots, n-2, n, n-1)$. Select edge $(\boldsymbol{s}^{(n-1)}, \boldsymbol{s})$.

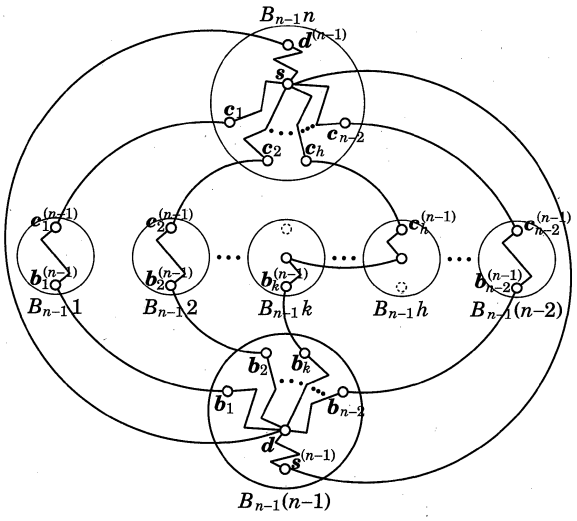


Fig. 7 Case V, Step 8.

Step 4 For each i ($1 \le i \le n-2$, $i \ne k$), select the shortest path from $\boldsymbol{s}$ to $\boldsymbol{c}_i = (1, 2, \ldots, i-1, i+1, \ldots, n-1, i, n)$ and edge $(\boldsymbol{c}_i, \boldsymbol{c}_i^{(n-1)})$.

Step 5 For each i ($1 \le i \le n-2$, $i \ne h$), select the shortest path from $\boldsymbol{d}$ to $\boldsymbol{b}_i = (d_1, d_2, \ldots, d_{l_i-1}, d_{l_i+1}, \ldots, d_{n-1}, i, d_n)$ where $d_{l_i} = i$, and select edge $(\boldsymbol{b}_i, \boldsymbol{b}_i^{(n-1)})$.

Step 6 For each i ($1 \le i \le n-2$, $i \ne h, k$), apply `shortest` in $B_{n-1}i$ to obtain the path from $\boldsymbol{c}_i^{(n-1)}$ to $\boldsymbol{b}_i^{(n-1)}$.

Step 7 If $h = k$, apply `shortest` in $B_{n-1}h$ to obtain the path from $\boldsymbol{c}_h^{(n-1)}$ to $\boldsymbol{b}_k^{(n-1)}$ and terminate the process.

Step 8 Apply `shortest` in $B_{n-1}h$ to obtain the path from $\boldsymbol{c}_h^{(n-1)}$ to $\tilde{\boldsymbol{c}}_h = (-, -, \cdots, -, k, h)$, where $-$ represents an arbitrary symbol other than k and h. Similarly, apply `shortest` in $B_{n-1}k$ to obtain the path from $\boldsymbol{b}_k^{(n-1)}$ to $\tilde{\boldsymbol{b}}_k = \tilde{\boldsymbol{c}}_h^{(n-1)}$. Select edge $(\tilde{\boldsymbol{c}}_h, \tilde{\boldsymbol{b}}_k)$.

Figure 7 shows the internally disjoint paths between $\boldsymbol{s}$ and $\boldsymbol{d}$ after Step 8 in Case V.

4. Proof of Correctness and Estimation of Complexity

Let $T(n)$ and $L(n)$ represent the time complexity of the proposed algorithm and the maximum path length obtained by it for an n-bubble sort graph, respectively. Then, we can prove the following theorem by induction on n.

Theorem 1: $n-1$ paths generated by the proposed algorithm are internally disjoint. The time complexity of their generation $T(n)$ is $O(n^4)$, and the maximum path length $L(n)$ is $n(n+1)/2$.

Proof : This is proved from the following Lemmas 1 to 5. □

Lemma 1: The $n-1$ paths generated by the Case I procedure are internally disjoint. The time complexity of the generation is $T(n-1) + O(n^3)$, and the maximum path length is $\max\{L(n-1), (n+1)(n-2)/2 + 3\}$.

Proof : Step 1 is recursive, therefore from the induction hypothesis, these $n-2$ paths are internally disjoint, and the time complexity and the maximum path length are $T(n-1)$ and $L(n-1)$, respectively. On the other hand, the path constructed by Steps 2 to 6 is outside the subgraph $B_{n-1}n$, except for its two terminal nodes. Therefore the path is internally disjoint with the other $n-2$ paths. Additionally, the path consists of subpaths constructed by two applications of the shortest path routing algorithm and three edges. The path generated in Step 3 has $n-2$ edges at most, while the path generated in Step 5 has $(n-1)(n-2)/2$ edges. Therefore, the time complexity for the path construction is $O(n^3)$ and the maximum path length is $(n+1)(n-2)/2 + 3$. □

Lemma 2: The $n-1$ paths generated by the Case II procedure are internally disjoint. The time complexity of generation is $O(n^4)$, and the maximum path length is $(n-1)(n+2)/2$.

Proof : The shortest path obtained in $B_{n-1}n$ for each i in Step 1 is obtained by moving only i. All nodes on the shortest path from $\boldsymbol{c}_{d_n}$ to $\boldsymbol{d}^{(n-1)}$ have the integer d_n at the semi final positions in their labels that make those nodes distinct from those on other shortest paths. Hence, it is clear that these paths are disjoint except for the source node $\boldsymbol{s}$. The $n-1$ edges selected in Step 2 are disjoint because one of the terminal nodes of each of the edges are different nodes in $B_{n-1}n$ and the other terminal nodes are in different subgraphs. Similar to Steps 1 and 2, the edges and paths obtained in Steps 4 and 5 are disjoint except for the destination node $\boldsymbol{d}$. Consequently, the $n-1$ paths obtained are internally disjoint.

In Step 1, the time complexity to construct the shortest subpath from $\boldsymbol{s}$ to $\boldsymbol{c}_{d_n}$ by `shortest` is $O(n^3)$. To construct each of the remaining $n-2$ subpaths in Step 1, `shortest` finds the element i and performs at most $n-2$ exchange operations in $O(n^2)$ time. Hence, the total time complexity of Step 1 is $O(n^3)$. The time complexity of Step 2 is $O(n^2)$ in total since it takes $O(n)$ to construct each edge. In Step 3, the shortest paths are obtained by `shortest` in $n-1$ subgraphs $B_{n-1}i$ ($1 \le i \le n-1$). Therefore, the time complexity is $O(n^4)$ in total. Similar to Step 2, the time complexity of Step 4 is $O(n^2)$. In Step 5, similar to the routing in Step 1, each shortest path from $\boldsymbol{b}_i$ to $\boldsymbol{d}$ is obtained by finding the element i in the label of $\boldsymbol{b}_i$ and at most $n-2$ exchange operations. Therefore, the total time complexity of Step 5 is $O(n^3)$.

The maximum path length given in Step 1 is attained by the subpath from $\boldsymbol{s}$ to $\boldsymbol{c}_{d_n}$, and its length is at most $(n-1)(n-2)/2$. However, $\boldsymbol{c}_{d_n}$ is identical to $\boldsymbol{d}^{(n)}$ that is a neighbor node of the destination node, and the subpath is not included in the total maximum path. The remaining $n-2$ subpaths generated in Step 1 may be included in the total maximum path. These subpaths are obtained by moving only one element i from $\boldsymbol{s}$. Hence, their maximum path length is $n-2$. The maximum path length given in Step 2 is 1. Because the $n-1$ subpaths given in Step 3 are the shortest paths between two nodes $\boldsymbol{c}_i^{(n-1)}$ and $\boldsymbol{b}_i^{(n-1)}$ in $B_{n-1}i$

$(1 \le i \le n-1)$, their maximum path length is $(n-1)(n-2)/2$. The maximum path length in Step 4 is 1. Similar to Step 1, the maximum path length in Step 5 is $n-2$. Consequently, the total maximum path is obtained by concatenating the subpaths whose lengths are $n-2$, 1, $(n-1)(n-2)/2$, 1, and $n-2$. Therefore, the maximum path length is $(n-1)(n+2)/2$. □

Lemma 3: The $n-1$ paths generated by the Case III procedure are internally disjoint. The time complexity of generation is $O(n^4)$, and the maximum path length is $(n-1)(n+2)/2$.

Proof: Similar to the Lemma 2 proof, it is possible to prove that the $n-1$ paths obtained are internally disjoint.

Each subpath in Step 1 is obtained by finding the element i and moving it to the appropriate position by at most $n-2$ exchange operations. Therefore, it takes $O(n^2)$ time complexity to construct each path. Hence, Step 1 takes $O(n^3)$ time complexity in total. In Step 2, since it takes $O(n)$ to construct each edge, $O(n^2)$ time complexity is required in total. In Step 3, the shortest paths are obtained in $n-1$ subgraphs $B_{n-1}i$ $(1 \le i \le n-1)$ by `shortest`. Therefore, the time complexity is $O(n^4)$ in total. Similar to Step 2, the time complexity of Step 4 is $O(n^2)$. In Step 5, the shortest path from $\boldsymbol{c}_{d_n}^{(n-1)}$ to $\boldsymbol{d}_n$ is constructed in subgraph $B_{n-1}d_n$. Hence, the time complexity is $O(n^3)$. Finally, similar to Step 1, the time complexity of Step 6 is $O(n^3)$.

Similar to the proof of Lemma 2, the maximum path lengths of Steps 1 to 4 and 6 are $n-2$, 1, $(n-1)(n-2)/2$, 1, and $n-2$, respectively. For all the nodes on the subpath generated in Step 5, the final two elements in their label are n and d_n in this order. Hence, the length of the subpath of Step 5 is at most $(n-2)(n-3)/2$. The candidate paths that may become the total maximum path are either the path that consists of the subpaths of Steps 1 to 4, and 6 or the path that consists of the subpaths of Steps 1, 2, 5, and 6. The lengths of the former and latter paths are at most $(n-2)+1+(n-1)(n-2)/2+1+(n-2) = (n-1)(n+2)/2$ and $(n-2)+1+(n-2)(n-3)/2+(n-2) = n(n-1)/2$, respectively. Therefore, the former gives the maximum path length. □

Lemma 4: The $n-1$ paths generated by the Case IV procedure are internally disjoint. The time complexity of generation is $O(n^4)$, and the maximum path length is $(n-1)(n+2)/2$.

Proof: Similar to the Lemma 2 proof, it is possible to prove that the $n-1$ paths obtained are internally disjoint. The time complexity of Step 1 is $O(n)$, and the path length obtained is 1. The time complexities of Steps 2 and 6 are both $O(n^3)$, and the maximum path lengths are both $n-2$. Additionally, the time complexities of Steps 3 and 5 are both $O(n^2)$, and only one edge is obtained in each step. Finally, the time complexity of Step 4 is $O(n^4)$, and the maximum path is $(n-1)(n-2)/2$. The total maximum path is attained by concatenating the subpaths generated in Steps 2 to 6 whose total length is $(n-1)(n+2)/2$. □

Lemma 5: The $n-1$ paths generated by the Case V procedure are internally disjoint. The time complexity of generation is $O(n^3)$, and the maximum path length is $n(n+1)/2$.

Proof: Step 2 constructs the path using the `shortest` algorithm, such that it first moves k to the right, then keeps the position of k to proceed to $\boldsymbol{d}^{(n-1)}$. However, Step 3 obtains the shortest path in $B_{n-1}n$ for each $i(\neq k)$ by moving only i to the right. Therefore, the paths obtained in Step 4 are disjoint, except for the source node $\boldsymbol{s}$ and they are disjoint with the path obtained in Step 2, except for $\boldsymbol{s}$. The $n-1$ edges obtained in Step 4 each have terminal node in $B_{n-1}n$, but they are all different nodes. The other nodes for these edges are in different subgraphs. Therefore, these edges are disjoint. The paths and edges obtained in Steps 3 and 5 are disjoint, except for the destination node, similar to the paths and edges obtained in Steps 2 and 4. The paths obtained in Step 6 are in different subgraphs and are disjoint. Moreover, all nodes included in the paths or the edges obtained in Steps 7 and 8 belong to subgraphs $B_{n-1}h$ or $B_{n-1}k$. Then, they are disjoint with the other paths. Consequently, the $n-1$ paths obtained are internally disjoint.

The time complexity of Step 1 is $O(n)$. The time complexities of Steps 2 to 5 are all $O(n^3)$. The time complexity of Step 6 is $O(n^4)$, and the time complexities of Steps 7 and 8 are both $O(n^3)$.

The path lengths of Step 2 and 3 are both $(n-2)(n-3)/2+1$. The maximum path lengths generated in Steps 4 and 5 are both $n-1$. The maximum path length of Step 6 is $(n-1)(n-2)/2$. The maximum path lengths of Steps 7 and 8 are $(n-1)(n-2)/2$ and $(n-1)(n-2)/2+1$, respectively. The lengths of the paths between $\boldsymbol{s}$ and $\boldsymbol{d}$ generated in Steps 1 and 2 are at most $(n-2)(n-3)/2+1$. Therefore, the total maximum path is attained by concatenating the subpaths generated by Steps 4, 5, and 8, and its length is $(n-1)+(n-1)+(n-1)(n-2)/2+1 = n(n+1)/2$. □

5. Experiment for Evaluation

To evaluate the average performance of the algorithm, we conducted a computer experiment by randomly sampling the destination node as follows:

1. For each $n = 3, 4, \ldots, 60$, repeat the following steps 100,000 times.
2. Fix the source node to $\boldsymbol{s} = (1, 2, \ldots, n)$.
3. Select the destination node $\boldsymbol{d}$ randomly other than $\boldsymbol{s}$.
4. For $\boldsymbol{s}$ and $\boldsymbol{d}$, apply the algorithm and measure the execution times and path lengths.

The algorithm was implemented using programming language C. The program was compiled using a `gcc` compiler with a `-O2` option. A target machine with an Intel Pentium IV, 2.4 GHz CPU, 256 MB main memory, and a Linux operating system was selected as the execution environment. The average execution times and the lengths of paths obtained by this experiment are shown in Figs. 8 and 9, respectively. From Fig. 8, we can see that the average execu-

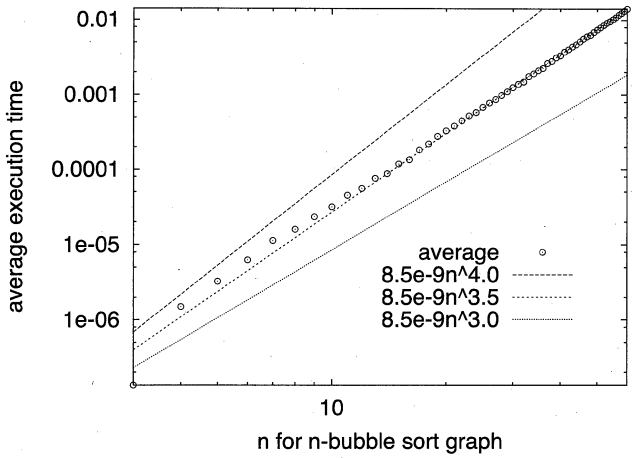


Fig. 8 Average execution time.

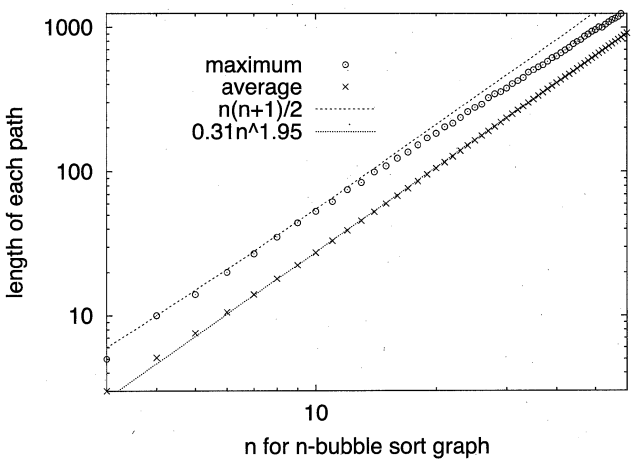


Fig. 9 Length of each path.

tion time is $O(n^{3.5})$. Also, from Fig. 9, we can see that the average length of each path is $O(n^{1.95})$.

6. Conclusion

In this paper, we have presented an algorithm that generates, for two arbitrary nodes in an n-bubble-sort graph, $n-1$ internally disjoint paths between those nodes. Moreover, we have proved the correctness of the algorithm and shown that its time complexity is $O(n^4)$, and the maximum path length is $n(n+1)/2$. Additionally, we have estimated the average performance of the algorithm by randomly sampling destination nodes. The results show that the average time complexity is $O(n^{3.5})$, and the average path length is $O(n^{1.95})$.

Future works include a more precise performance evaluations of the algorithm, and performance improvement with respect to the time complexity and path lengths. We will also apply our approach to a modified bubble-sort graph, which is obtained by relaxing the exchange operation to permit the first and last symbols.

Acknowledgment

We really appreciate the anonymous reviewers for their constructive suggestions and insightful comments.

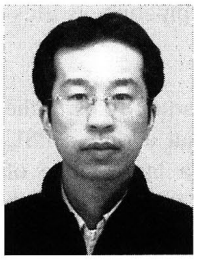
This study is partly supported by a Fund of the Ministry of Education, Culture, Sports, Science and Technology Japan for Promoting Research on Symbiotic Information Technology. It is also partly supported by a Grant-in-Aid for Scientific Research (C) of the Japan Society for the Promotion of Science under Grant No. 19500022.

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] F.S. Annexstein and M. Baumslag, "A unified approach to off-line permutation routing on parallel networks," Proc. 2nd Annual ACM Symposium on Parallel Algorithms Architectures, pp.398–406, July 1990.
[3] J.C. Bermond and C. Peyrat, "De Bruijn and Kautz networks — A competitor for the hypercube?," Proc. 1st European Workshop on Hypercubes and Distributed Computers, pp.279–293, Oct. 1989.
[4] 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.
[5] M. Dietzfelbinger, S. Madhavapeddy, and I.H. Sudborough, "Three disjoint path paradigms in star networks," Proc. IEEE Symposium on Parallel and Distributed Processing, pp.400–406, Dec. 1991.
[6] W.H. Gates and C.H. Papadimitriou, "Bounds for sorting by prefix reversal," Discrete Mathematics, vol.27, pp.47–57, 1979.
[7] 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.
[8] 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.
[9] 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.
[10] 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.
[11] D.E. Knuth, Axioms, and Hulls, Lecture Notes in Computer Science, vol.606, Springer, 1992.
[12] D.E. Knuth, The Art of Computer Programming, Volume 3: Sorting and Searching, 2nd Edition, Addison-Wesley, 1998.
[13] S. Lakshmivarahan, J.S. Jwo, and S.K. Dhall, "Symmetry in interconnection networks based on cayley graphs of permutation groups: A survey," Parallel Comput., vol.19, no.4, pp.361–407, April 1993.
[14] 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, Feb. 1996.
[15] 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, Dec. 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, 1989.
[17] C.L. Seitz, "The cosmic cube," Commun. ACM, vol.28, no.1, pp.22–33, Jan. 1985.
[18] Y. Suzuki and K. Kaneko, "An algorithm for node-disjoint paths in pancake graphs," IEICE Trans. Inf. & Syst., vol.E86-D, no.3, pp.610–615, March 2003.

[19] Y. Suzuki and K. Kaneko, "An algorithm for disjoint paths in bubble-sort graphs," Systems and Computers in Japan, vol.37, no.12, pp.27–32, Nov. 2006.

[20] C.H. Yeh and E.A. Varvarigos, "Macro-star networks — Efficient low-degree alternatives to star graphs," IEEE Trans. Parallel Distrib. Syst., vol.9, no.10, pp.987–1003, Oct. 1998.



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

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.