A Parallel Algorithm for Constructing Multiple Independent Spanning Trees in Bubble-Sort Networks
Shih-Shun Kao, Ralf Klasing, Ling-Ju Hung, Sun-Yuan Hsieh

To cite this version:
Shih-Shun Kao, Ralf Klasing, Ling-Ju Hung, Sun-Yuan Hsieh. A Parallel Algorithm for Constructing Multiple Independent Spanning Trees in Bubble-Sort Networks. AAIM 2021, Dec 2021, Dallas, United States. pp.252-264, 10.1007/978-3-030-93176-6_22 . hal-03820079

HAL Id: hal-03820079
https://hal.science/hal-03820079
Submitted on 21 Oct 2022

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L’archive ouverte pluridisciplinaire HAL, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d’enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.
A parallel algorithm for constructing multiple independent spanning trees in bubble-sort networks

Shih-Shun Kao¹,², Ralf Klasing¹, Ling-Ju Hung³, and Sun-Yuan Hsieh²,⁴

¹ CNRS, LaBRI, Université de Bordeaux, 351 Cours de la Libération, 33405 Talence, France {shih-shun.kao, ralf.klasing}@labri.fr
² Department of Computer Science and Information Engineering, National Cheng Kung University, No. 1, University Road, Tainan, Taiwan
³ Department of Creative Technologies and Product Design, National Taipei University of Business, No.100, Sec. 1, Fulong Road, Taoyuan, Taiwan
⁴ Institute of Medical Informatics, National Cheng Kung University, No. 1, University Road, Tainan, Taiwan

Abstract. The use of multiple independent spanning trees (ISTs) for data broadcasting in networks provides a number of advantages, including the increase of fault-tolerance and secure message distribution. Thus, the designs of multiple ISTs on several classes of networks have been widely investigated. Kao et al. [Journal of Combinatorial Optimization 38 (2019) 972-986] proposed an algorithm to construct independent spanning trees in bubble-sort networks. The algorithm is executed in a recursive function and thus is hard to parallelize. In this paper, we focus on the problem of constructing ISTs in bubble-sort networks \( B_n \) and present a non-recursive algorithm. Our approach can be fully parallelized, i.e., every vertex can determine its parent in each spanning tree in constant time. This solves the open problem from the paper by Kao et al. Furthermore, we show that the total time complexity \( \mathcal{O}(n \cdot n!) \) of our algorithm is asymptotically optimal, where \( n \) is the dimension of \( B_n \) and \( n! \) is the number of vertices of the network.

Keywords: Independent spanning trees · Bubble-sort networks · Interconnection networks.

1 Introduction

The design of modern interconnected networks faces several critical demands, such as how to perform fault-tolerant transmission and secure message distribution in a reliable communication network. The practical solution to meet the

* This research was supported by the LabRI under the “Projets émergents” program. This study has been carried out in the frame of the “Investments for the future” Programme IdEx Bordeaux - SysNum (ANR-10-IDEX-03-02).
above requirements is to design a multi-path routing mechanism, which requires the network to provide disjoint paths between each pair of vertices. Therefore, if the transmission fails due to a disconnection in the current transmission path, we can resume the data transmission via another disjoint backup path. This dramatically increases the performance of fault-tolerant communication [2, 14]. In addition, disjoint paths could be used in secure message distribution over a fault-free network in the following way [2, 30]. A message can be divided into several packets where the source node sends each packet to its destination via different paths. Thus, each node in the network receives at most one of the packets except for the destination node that receives all the packets.

Usually, an interconnection network is modeled by a simple undirected graph \( G = (V, E) \), where the vertex set \( V(G) \) and the edge set \( E(G) \) represent the set of processors and the set of communication links between the processors, respectively. A spanning tree \( T \) in \( G \) is a connected acyclic subgraph of \( G \) such that \( V(T) = V(G) \). Two spanning trees rooted at a specific vertex, say \( r \), are called independent spanning trees (ISTs for short) if, for any vertex \( v \in V(G) \setminus \{r\} \), the two paths from \( v \) to \( r \) in any two trees share no common edge and no common vertex except for \( v \) and \( r \). Accordingly, the provision of multiple ISTs suffices to meet the requirement of reliable communication in a network.

Research on ISTs has been conducted for nearly three decades. In 1989, Zehavi and Itai [40] conjectured that there exist \( k \) ISTs rooted at an arbitrary vertex in a \( k \)-connected graph. From then on, this conjecture has been confirmed only for \( k \)-connected graphs with \( k \leq 4 \) (see [9, 10, 14]). Since this conjecture is still unsolved for general \( k \)-connected graphs for \( k \geq 5 \), the follow-up research mainly focused on the study of constructing ISTs on specific interconnection networks, e.g., the construction of ISTs on some variations of hypercubes [3, 20, 29, 30, 37], torus networks [28], recursive circulant graphs [34, 35], and special subclasses of Cayley networks [7, 8, 12, 13, 15, 19, 39]. In particular, special topics related to ISTs include the research on reducing the height of the ISTs [31, 33, 36] and parallel construction of ISTs [4–6, 32, 37, 38].

Note that there is a similar problem called the construction of completely independent spanning trees (CISTs for short) in a network. A set of \( k \) unrooted spanning trees are called CISTs if they are pairwise edge-disjoint and inner-node-disjoint (i.e., for each pair of vertices \( u \) and \( v \) in any two spanning trees, there exist no common edge and vertex in the paths between \( u \) and \( v \) except for the two end vertices). In particular, if \( k = 2 \), the two CISTs are called a dual-CIST. Hasunuma [11] showed that the problem of determining whether there exists a dual-CIST in a graph is NP-complete. He also conjectured that there exist \( k \) CISTs in a \( 2k \)-connected graph. Currently, this conjecture has been proved to fail by counterexamples [21, 26]. For recent research results on CISTs and their applications, the reader is referred to [22–25] and references quoted therein. Here, we explicitly point out that the construction of multiple ISTs and CIST are two different problems.

For the construction of ISTs on bubble-sort networks, Kao et al. [15] proposed an algorithm to construct \( n - 1 \) ISTs of \( B_n \) and showed that the algorithm has
optimal amortized efficiency for multiple trees construction. In particular, every vertex can determine its parent in each spanning tree in constant amortized time. The algorithm is executed in a recursive function and thus is hard to parallelize. In this paper, we present a parallel algorithm to construct \( n - 1 \) ISTs in bubble-sort networks \( B_n \). Our approach can be fully parallelized, i.e., every vertex can determine its parent in each spanning tree in constant time. This solves the open problem from [15]. Furthermore, we show that the total time complexity \( O(n \cdot n!) \) of our algorithm is asymptotically optimal, where \( n \) is the dimension of \( B_n \) and \( n! \) is the number of vertices of the network.

The rest of this paper is organized as follows. In Section 2, we introduce the bubble-sort graphs and some notations. In Section 3, we introduce the algorithm for constructing independent spanning trees of \( B_n \). In Section 4, we show the correctness of our algorithm and give the complexity analysis. Finally, conclusions and future works are given in Section 5.

2 Preliminaries

Let \( \Sigma_n \) be the set of all permutations on \( \{1, 2, \ldots, n\} \). For a permutation \( p \in \Sigma_n \) and an integer \( i \in \{1, 2, \ldots, n\} \), we use the following notations. The symbol at the \( i \)th position of \( p \) is denoted by \( p_i \), and the position where the symbol \( i \) appears in \( p \) is denoted by \( p^{-1}(i) \). A symbol \( i \) is said to be at the right position of \( p \) if \( p_i = i \), and for \( p \neq 1 \cdots n \) the position of the first symbol \( i \) from the right which is not in the right position is denoted by \( r(p) \). For \( i \in \{1, \cdots, n-1\} \), let \( p(i) = p_ip_{i+1} \cdots p_{n-1} \) be the permutation of \( \Sigma_n \) obtained from \( p \) by swapping two consecutive symbols at positions \( i \) and \( i+1 \). The bubble-sort network, denoted by \( B_n \), is an undirected graph consisting of the vertex set \( V(B_n) = \Sigma_n \) and the edge set \( E(B_n) = \{(x, x(i)) : x \in \Sigma_n, 1 \leq i \leq n-1\} \), where the edge \( (x, x(i)) \) is called an \( i \)-edge of \( B_n \). Thus, \( B_n \) is a Cayley graph generated by the transposition set \( \{(i, i+1) : 1 \leq i \leq n-1\} \), which is specified by an \( n \)-path \( P_n = (1, 2, \ldots, n) \) as its transposition graph [1, 16]. For example, Fig. 1 depicts \( B_4 \). Clearly, for \( B_n \), the transposition graph \( P_n \) contains only two subgraphs isomorphic to an \( (n-1) \)-path: one is \((1, 2, \ldots, n-2)\) and the other is \((2, 3, \ldots, n-1)\). Thus, for \( n \geq 3 \), there are exactly two ways to decompose \( B_n \) into \( n \) disjoint subgraphs that are isomorphic to \( B_{n-1} \). Let \( B^1_n \) denote the graph obtained from \( B_n \) by removing the set of all \( i \)-edges. Then, both \( B^1_n \) and \( B^{n-1}_n \) consist of \( n \) disjoint subgraphs isomorphic to \( B_{n-1} \).
The bubble-sort network $B_4$

Fig. 1. The bubble-sort network $B_4$

a spanning tree, for each vertex $v = v_1 \cdots v_n \in V(B_n)$ except the root $1_n$, we denote by $\text{Parent}(v, t, n)$ the parent of $v$ in $T^n_t$.

The case $n = 3$. Since $B_3$ is isomorphic to a 6-cycle, we have

$$\begin{align*}
\text{Parent}(v, 1, 3) = & \begin{cases} 
123 & \text{if } v = 213; \\
213 & \text{if } v = 231; \\
231 & \text{if } v = 321; \\
321 & \text{if } v = 312; \\
312 & \text{if } v = 132; \\
132 & \text{if } v = 123.
\end{cases} \\
\text{and Parent}(v, 2, 3) = & \begin{cases} 
231 & \text{if } v = 213; \\
321 & \text{if } v = 231; \\
312 & \text{if } v = 321; \\
132 & \text{if } v = 312; \\
123 & \text{if } v = 132.
\end{cases}
\end{align*}$$

That is, the two paths $T^n_1 = (132, 312, 321, 231, 213, 123)$ and $T^n_2 = (213, 231, 321, 312, 132, 123)$ are ISTs of $B_3$ that take $1_3 = 123$ as the common root.

The case $n \geq 4$. In general, for $B_n$ with $n \geq 4$, the construction of the ISTs of $B_n$ can be accomplished by Algorithm 1 to determine the parent of each vertex (except the root) in every spanning tree.

The main idea of the algorithm is as follows. In $T^n_t$ for $t \in \{1, 2, \ldots, n - 2\}$ all paths are from the vertex $x$ with $x_n \in \{1, 2, \ldots, n - 1\} \setminus \{t\}$ to the vertex $y$ with $y_n = t$. Then, all paths are from the vertex $y$ with $y_n = t$ to the root $r$. In $T^n_{n-1}$ all paths are from the vertex $v$ with $v_n = n$ to the vertex $u$ with $u_n \in \{1, 2, \ldots, n - 1\}$. Then, all paths are from the vertex $u$ with $u_n \in \{1, 2, \ldots, n - 1\}$ to the root $r$. 

Algorithm 1: The new parallel algorithm

Input: v: the vertex $v = v_1 \cdots v_n$ in $B_n$
      t: the t-th tree $T^n_t$ in IST
      n: the dimension of $B_n$

Output: p: $p = \text{Parent}(v, t, n)$ the parent of $v$ in $T^n_t$

1 if $v_n = n$ then
2     if $t = 2$ and $\text{Swap}(v, t) = 1_n$ then $p = \text{Swap}(v, t - 1)$
3     else if $t = n - 1$ then $p = \text{Swap}(v, v_{n-1})$
4         else $p = \text{FindPosition}(v)$
5     end
6 else
7     if $v_n = n - 1$ and $v_{n-1} = n$ and $\text{Swap}(v, n) \neq 1_n$ then
8         if $t = 1$ then $p = \text{Swap}(v, n)$
9             else $p = \text{Swap}(v, t - 1)$
10        end
11 else
12     if $v_n = t$ then $p = \text{Swap}(v, n)$
13         else $p = \text{Swap}(v, t)$
14     end
15 end
16 return $p$

Function FindPosition(v)

Input: v: the vertex $v = v_1 \cdots v_n$ in $B_n$

Output: p: $p = \text{Parent}(v, t, n)$ the parent of $v$ in $T^n_t$

1 if $v_{n-1} \in \{t, n - 1\}$ then $j = r(v), p = \text{Swap}(v, v_j)$
2 else $p = \text{Swap}(v, t)$
3 return $p$

Function Swap(v, x)

Input: v: the vertex $v = v_1 \cdots v_n$ in $B_n$
       x: the symbol in the vertex $v_1 \cdots v_n$

Output: p: $p = \text{Parent}(v, t, n)$ the parent of $v$ in $T^n_t$

1 $i = v^{-1}(x), p = v(i)$
2 return $p$
Table 1. The parent of every vertex \( v \in V(B_4) \setminus \{1_4\} \) in \( T_t^4 \) for \( t \in \{1, 2, 3\} \) calculated by Algorithm 1.

<table>
<thead>
<tr>
<th>( v )</th>
<th>( v_4 )</th>
<th>( p )</th>
<th>( v )</th>
<th>( v_4 )</th>
<th>( p )</th>
</tr>
</thead>
<tbody>
<tr>
<td>1234</td>
<td>-</td>
<td>3</td>
<td>3124</td>
<td>1</td>
<td>3214</td>
</tr>
<tr>
<td>1243</td>
<td>2</td>
<td>1</td>
<td>2134</td>
<td>3</td>
<td>3142</td>
</tr>
<tr>
<td>1234</td>
<td>3</td>
<td>3</td>
<td>1234</td>
<td>3</td>
<td>3142</td>
</tr>
<tr>
<td>1324</td>
<td>2</td>
<td>4</td>
<td>1234</td>
<td>2</td>
<td>3124</td>
</tr>
<tr>
<td>3124</td>
<td>3</td>
<td>3</td>
<td>1342</td>
<td>3</td>
<td>3241</td>
</tr>
<tr>
<td>1342</td>
<td>2</td>
<td>2</td>
<td>1324</td>
<td>1</td>
<td>3214</td>
</tr>
<tr>
<td>1342</td>
<td>3</td>
<td>3</td>
<td>1342</td>
<td>3</td>
<td>3241</td>
</tr>
<tr>
<td>1423</td>
<td>3</td>
<td>1</td>
<td>4132</td>
<td>1</td>
<td>2341</td>
</tr>
<tr>
<td>1432</td>
<td>2</td>
<td>1</td>
<td>4321</td>
<td>2</td>
<td>1423</td>
</tr>
<tr>
<td>1432</td>
<td>3</td>
<td>3</td>
<td>4321</td>
<td>3</td>
<td>4321</td>
</tr>
<tr>
<td>2134</td>
<td>3</td>
<td>1</td>
<td>3124</td>
<td>1</td>
<td>2314</td>
</tr>
<tr>
<td>2143</td>
<td>2</td>
<td>1</td>
<td>2134</td>
<td>1</td>
<td>3241</td>
</tr>
<tr>
<td>2143</td>
<td>3</td>
<td>3</td>
<td>2143</td>
<td>3</td>
<td>4321</td>
</tr>
<tr>
<td>2341</td>
<td>2</td>
<td>1</td>
<td>3241</td>
<td>1</td>
<td>2341</td>
</tr>
<tr>
<td>2341</td>
<td>3</td>
<td>3</td>
<td>2341</td>
<td>3</td>
<td>3421</td>
</tr>
<tr>
<td>2413</td>
<td>3</td>
<td>1</td>
<td>4132</td>
<td>1</td>
<td>3421</td>
</tr>
<tr>
<td>2413</td>
<td>3</td>
<td>3</td>
<td>2413</td>
<td>3</td>
<td>4321</td>
</tr>
</tbody>
</table>

Note that in a pre-processing stage, each node \( v = v_1v_2 \cdots v_n \) (\( v \neq 1_n \)) computes its inverse permutation, i.e., \( v^{-1}(1)v^{-1}(2) \cdots v^{-1}(n) \), and the position of the first symbol \( i \) from the right which is not in the right position, i.e., \( r(v) \). This can be done efficiently in \( O(n) \) time for each vertex. Algorithm 1 uses two functions \( \text{FindPosition}(v) \) and \( \text{Swap}(v, x) \). The function \( \text{FindPosition}(v) \) finds the rightmost symbol \( x \) in \( v \) which is not in the right position, and then calls the \( \text{Swap}(v, x) \) function. The function \( \text{Swap}(v, x) \) swaps the symbol \( x \) in \( v \) in its position \( i \) with the symbol in position \( i + 1 \). Since we have the pre-processing stage, the two functions \( \text{FindPosition}(v) \) and \( \text{Swap}(v, x) \) can be calculated in constant time.

Table 1 shows the parent of every vertex \( v \in V(B_4) \setminus \{1_4\} \) in \( T_t^4 \) for \( t \in \{1, 2, 3\} \) calculated by Algorithm 1. For example, we consider \( v = 3214 \) and \( t = 3 \). Since \( v_4 = 4, p = \text{Swap}(v, v_{4-1}) = 3241 \). Also, we consider \( v = 4321 \) and \( t = 1 \). Since \( v_4 = 1, p = \text{Swap}(v, 4) = 4321 \). The corresponding three ISTs rooted at vertex \( 1_4 \) for \( B_4 \) are shown in Fig. 2.
4 Correctness and complexity analysis

In this section, we first show the correctness of Algorithm 1. Let $T$ be a tree and $u, v \in V(T)$, we use $T(u, v)$ to denote the unique path joining $u$ and $v$ in $T$. For two spanning trees $T^n_t$ and $T^n_{t'}$ for $t, t' \in \{1, 2, \ldots, n-1\}$ with $t \neq t'$, we denote by $T^n_t(v, r)$ and $T^n_{t'}(v, r)$ the two paths from $v$ to the common $r$.

**Theorem 1.** For $n \geq 4$, $T^n_1, T^n_2, \ldots, T^n_{n-1}$ are $n-1$ ISTs of $B_n$.

**Proof.** Suppose that $n \geq 4$, let $r = 1_n (= 12 \cdots n)$, the proof is by showing that for any vertex $v \in V(B_n) \setminus \{r\}$, the two paths from $v$ to $r$ in any two trees of $T^n_1, T^n_2, \ldots, T^n_{n-1}$ share no common edge and no common vertex except for $v$ and $r$, and thereby proving the independence. Consider the following three cases:

**Case 1:** $v_n = n$.

Each vertex of the two paths $T^n_t(v, r)$ and $T^n_{t'}(v, r)$ (apart from $T^n_{n-1}(v, r)$) swaps symbol $t$ (resp., $t'$) to the position $v_{n-1}$ for $t, t' \in \{1, 2, \ldots, n-2\}$. Then, the rightmost symbol $i$ which is not in the right position swaps to the right position. Therefore, $T^n_t(v, r)$ and $T^n_{t'}(v, r)$ are vertex-disjoint. Now consider $T^n_{n-1}(v, r)$, each vertex of the path swaps the position $v_{n-1}$ to $v_n$. Then, the vertex $v$ with $v_n = n$ swaps the symbol $n-1$ to the position $v_n$. Hence, $T^n_t(v, r)$, $T^n_{t'}(v, r)$ and $T^n_{n-1}(v, r)$ are vertex-disjoint. See Fig. 3, the paths from the vertex $v$ with $v_n = n$ to $r$ are marked in red, in $T^n_{n-1}(v, r)$ each vertex of the path has symbol $n-1$ in $v_n$. The other trees $T^n_t(v, r)$ have symbol $t$ in position $v_n$ for $t \in \{1, 2, \ldots, n-2\}$.
Fig. 3. An illustration of the paths described in the proof of Case 1 of Theorem 1

Fig. 4. An illustration of the paths described in the proof of Case 2 of Theorem 1

Fig. 5. An illustration of the paths described in the proof of Case 3 of Theorem 1
Case 2: \(v_n = n - 1\).

Each vertex of the two paths \(T^n_i(v, r)\) and \(T^n_p(v, r)\) (apart from \(T^n_{n-1}(v, r)\)) swaps symbol \(t\) (resp., \(t'\)) to the position \(v_n\) for \(t, t' \in \{1, 2, \ldots, n - 2\}\). On the other hand each vertex of the path has symbol \(t\) (resp., \(t'\)) in different position. Therefore, \(T^n_i(v, r)\) and \(T^n_p(v, r)\) are vertex-disjoint. In \(T^n_{n-1}(v, r)\) each vertex of the path swaps symbol \(n\) to the position \(v_n\). By Case 1, the paths \(T^n_i(v, r)\) and \(T^n_{n-2}(v, r)\) are vertex-disjoint. Hence, \(T^n_i(v, r)\), \(T^n_p(v, r)\) and \(T^n_{n-1}(v, r)\) are vertex-disjoint. See Fig. 4, the paths from the vertex \(v\) with \(v_n = n - 1\) to \(r\) are marked in red, in \(T^n_i(v, r)\) each vertex of the path has symbol \(n - 1, n - 2\) or \(n\) in the position \(v_n\), in \(T^n_{n-1}(v, r)\) each vertex of the path swaps symbol \(n\) to the position \(v_n\).

Case 3: \(v_n = j\) for \(j \in \{1, 2, \ldots, n - 2\}\).

Each vertex of the two paths \(T^n_i(v, r)\) and \(T^n_p(v, r)\) (apart from \(T^n_{n-1}(v, r)\)) swaps symbol \(t\) (resp., \(t'\)) to the position \(v_n\) for \(t, t' \in \{1, 2, \ldots, n - 2\}\). On the other hand each vertex of the path has symbol \(t\) in different position. Therefore, \(T^n_i(v, r)\) and \(T^n_p(v, r)\) are vertex-disjoint. In \(T^n_{n-1}(v, r)\) each vertex of the path swaps symbol \(n - 1\) to \(v_n\). By Case 2, the paths \(T^n_i(v, r)\) and \(T^n_{n-2}(v, r)\) are vertex-disjoint. Hence, \(T^n_i(v, r)\), \(T^n_p(v, r)\) and \(T^n_{n-1}(v, r)\) are vertex-disjoint. See Fig. 5, the paths from the vertex \(v\) with \(v_n = 1\) to \(r\) are marked in red, in \(T^n_i(v, r)\) each vertex of the path swaps symbol \(n\) to \(v_n\), in \(T^n_{n-2}(v, r)\) each vertex of the path swaps symbol \(n - 2\) or \(n\) to \(v_n\), in \(T^n_{n-1}(v, r)\) each vertex of the path swaps symbol \(n - 1\) to \(v_n\). This completes the proof.

The height of a rooted tree \(T\), denoted by \(h(T)\), is the number of edges from the root to a farthest leaf. We define \(H_n = \max_{1 \leq i \leq n-1} h(T^n_i)\) to analyze the height of our constructed ISTs for \(B_n\).

**Theorem 2.** For the bubble-sort graph \(B_n\), Algorithm 1 correctly constructs \(n - 1\) ISTs of \(B_n\) with height at most \(n(n + 1)/2 - 1\). In particular, every vertex can determine its parent in each spanning tree in constant time.

**Proof.** From Algorithm 1, the path from the vertex \(v\) with \(v_n = 2\) to the vertex \(u\) with \(u_n = 1\) has at most \(n - 1\) edges, and the path from the vertex \(u\) with \(u_n = 1\) to the vertex \(x\) with \(x_n = n\) has at most \(n - 1\) edges. Moreover, the path from the vertex \(w\) with \(w_n = n\) to the vertex \(x\) with \(x_n = n\) and \(x_{n-1} = t\) has at most \(n - 1\) edges, and the path from the vertex \(x\) with \(x_n = n\) and \(x_{n-1} = t\) to the vertex \(y\) with \(y_n = n\) and \(y_{n-1} = n - 1\) has at most \(n - 2\) edges, and the path from the vertex \(y\) with \(y_n = n\) and \(y_{n-1} = n - 1\) to the vertex \(z\) with \(z_n = n\), \(z_{n-1} = n - 1\) and \(z_{n-2} = n - 2\) has at most \(n - 3\) edges. Since \((n - 2) + (n - 3) + \cdots + 1 = (n - 1)(n - 2)/2\), the path from the vertex \(x\) with \(x_n = n\) and \(x_{n-1} = t\) to the root \(r\) has at most \((n - 1)(n - 2)/2\) edges. The path from the vertex \(v\) with \(v_n = 2\) to the root \(r\) has at most \((n - 1)(n - 2)/2 + (n - 1) + (n - 1) = (n^2 - 3n + 2 + 4n - 4)/2 = (n^2 + n - 2)/2 = n(n + 1)/2 - 1\) edges. Hence, \(H_n \leq n(n + 1)/2 - 1\). Obviously, each vertex in Algorithm 1 can determine its parent in each spanning tree in constant time. This completes the proof.
Corollary 1. The total time complexity $O(n \cdot n!)$ of Algorithm 1 is asymptotically optimal.

Proof. There are $n - 1$ ISTs, each IST contains $n!$ vertices, hence the lower bound $\Omega(n \cdot n!)$ is obvious. Since each vertex in Algorithm 1 can determine its parent in each spanning tree in constant time, the total time complexity of the proposed Algorithm 1 is $O(n \cdot n!)$. Hence, the total time complexity $O(n \cdot n!)$ of Algorithm 1 is asymptotically optimal. This completes the proof. □

5 Conclusion

In this paper, we have proposed an algorithm for constructing $n - 1$ ISTs rooted at an arbitrary vertex of the bubble-sort network $B_n$. Our approach can be fully parallelized, i.e., every vertex can determine its parent in each spanning tree in constant time. Furthermore, we show that the total time complexity $O(n \cdot n!)$ of our algorithm is asymptotically optimal, where $n$ is the dimension of $B_n$ and $n!$ is the number of vertices of the network.

Since $B_n$ is a regular graph with connectivity $n-1$, the number of constructed ISTs is the maximum possible. For future work, a problem remaining open from our work is whether our algorithm can be extended to the $(n,k)$-bubble-sort graph [27, 41, 42] which is a generalization of bubble-sort networks. Moreover, the butterfly graph [17, 18] has good structural symmetries, is regular of degree 4, and the recursive construction properties are similar to bubble-sort networks. Thus, it is of interest to study the construction of ISTs on butterfly graphs.

References

A parallel algorithm for constructing ISTs in Bubble-Sort Networks


