A Parallel Algorithm for Constructing Multiple Independent Spanning Trees in Bubble-Sort Networks

Shih-Shun Kao, Ralf Klasing, Ling-Ju Hung, Sun-Yuan Hsieh

A parallel algorithm for constructing multiple independent spanning trees in bubble-sort networks*

Shih-Shun Kao[1,2], Ralf Klasing[1], Ling-Ju Hung[3], and Sun-Yuan Hsieh[2,4]

[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 ljhung@ntub.edu.tw

[4] Institute of Medical Informatics, National Cheng Kung University, No. 1, University Road, Tainan, Taiwan hsiehsy@mail.ncku.edu.tw

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 $\mathcal{O}(n{\cdot}n!)$ of our algorithm is asymptotically optimal, where n is the dimension of B_n and $n!$ is the number of vertices of the network.

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 $\varSigma_n$ be the set of all permutations on $\{1,2,\ldots,n\}$. For a permutation $p \in \varSigma_n$ and an integer $i \in \{1,2,\ldots,n\}$, we use the following notations. The symbol at the ith 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 $p \neq 12\cdots n$ 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 $p\langle i\rangle = p_1p_2\cdots p_{i-1}p_{i+1}p_ip_{i+2}\cdots p_n$ be the permutation of $\varSigma_n$ 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 $V(B_n) = \varSigma_n$ and the edge set $E(B_n) = \{(\mathbf{x},\mathbf{x}\langle i\rangle)\colon\ \mathbf{x} \in \varSigma_n, 1 \leqslant i \leqslant n-1\}$, where the edge $(\mathbf{x},\mathbf{x}\langle i\rangle)$ is called an *i-edge* of B_n. Thus, B_n is a Cayley graph generated by the transposition set $\{(i,i+1)\colon\ 1 \leqslant i \leqslant n-1\}$, 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 $n \geqslant 3$, there are exactly two ways to decompose B_n into n disjoint subgraphs that are isomorphic to B_{n-1}. Let B_n^i denote the graph obtained from B_n by removing the set of all i-edges. Then, both B_n^1 and B_n^{n-1} consist of n disjoint subgraphs isomorphic to B_{n-1}.

3 Constructing ISTs on B_n

In this section, we present an algorithm for constructing $n-1$ ISTs of B_n. Since B_n is vertex-transitive, without loss of generality, we may choose the identity $\mathbf{1}_n = 12\cdots n$ as the common root of all ISTs. Also, since B_n has connectivity $n-1$, the root in every spanning tree has a unique child. For $1 \leqslant t \leqslant n-1$, if the root of a spanning tree takes $\mathbf{1}_n\langle t\rangle = 12\cdots(t-1)(t+1)t(t+2)\cdots n$ as its unique child, then the spanning tree of B_n is denoted by T_t^n. To describe such

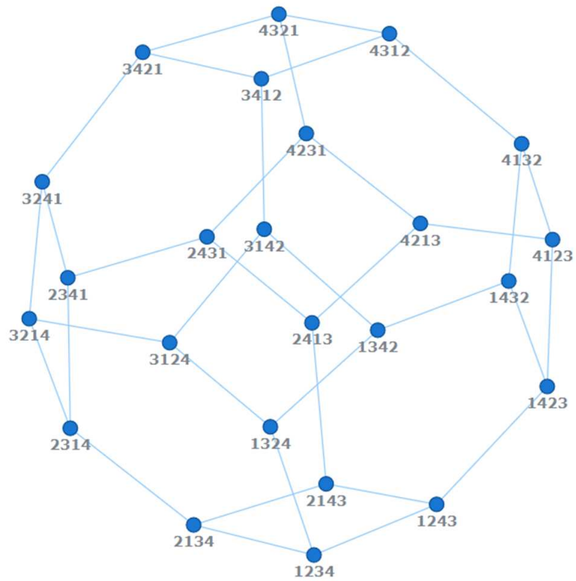

Fig. 1. The bubble-sort network B_4

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

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

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

That is, the two paths $T_1^3 = (132, 312, 321, 231, 213, 123)$ and $T_2^3 = (213, 231, 321, 312, 132, 123)$ are ISTs of B_3 that take $\mathbf{1}_3 = 123$ as the common root.

The case $n \geqslant 4$. In general, for B_n with $n \geqslant 4$, 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_t^n 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-1}^n 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_t^n in IST
n: the dimension of B_n
Output: p: $p = \text{Parent}(v, t, n)$ the parent of v in T_t^n

```
if v_n = n then
    if t = 2 and Swap(v, t) = 1_n then p = Swap(v, t − 1)
    else if t = n − 1 then p = Swap(v, v_{n−1})
    else p = FindPosition(v)
end
else
    if v_n = n − 1 and v_{n−1} = n and Swap(v, n) ≠ 1_n then
        if t = 1 then p = Swap(v, n)
        else p = Swap(v, t − 1)
    end
    else
        if v_n = t then p = Swap(v, n)
        else p = Swap(v, t)
    end
end
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_t^n

```
if v_{n−1} ∈ {t, n − 1} then j = r(v), p = Swap(v, v_j)
else p = Swap(v, t)
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_t^n

```
i = v^{−1}(x), p = v⟨i⟩
return p
```

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

v	t	v_4	p	v	t	v_4	p
					1		3214
1234	-	-	-	3124	2	4	1324
					3		3142
	1		2143		1		3412
1243	2	3	1423	3142	2	2	3124
	3		1234		3		1342
	1		3124		1		2314
1324	2	4	1234	3214	2	4	3124
	3		1342		3		3241
	1		3142		1		3214
1342	2	2	1324	3241	2	1	3421
	3		1432		3		2341
	1		4123		1		3421
1423	2	3	1432	3412	2	2	3142
	3		1243		3		4312
	1		4132		1		3241
1432	2	2	1342	3421	2	1	3412
	3		1423		3		4321
	1		1234		1		4213
2134	2	4	2314	4123	2	3	4132
	3		2143		3		1423
	1		2134		1		4312
2143	2	3	2413	4132	2	2	1432
	3		1243		3		4123
	1		2134		1		4231
2314	2	4	3214	4213	2	3	4123
	3		2341		3		2413
	1		2314		1		2431
2341	2	1	3241	4231	2	1	4321
	3		2431		3		4213
	1		2431		1		4321
2413	2	3	4213	4312	2	2	3412
	3		2143		3		4132
	1		2341		1		3421
2431	2	1	4231	4321	2	1	4312
	3		2413		3		4231

Note that in a pre-processing stage, each node $v = v_1 v_2 \cdots v_n$ ($v \neq \mathbf{1}_n$) 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 $\mathcal{O}(n)$ time for each vertex. Algorithm 1 uses two functions FindPosition(v) and Swap(v, x). The function FindPosition(v) finds the rightmost symbol x in v which is not in the right position, and then calls the Swap(v, x) function. The function 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 FindPosition(v) and Swap(v, x) can be calculated in constant time.

Table 1 shows the parent of every vertex $v \in V(B_4) \setminus \{\mathbf{1}_4\}$ 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 $v_4 = 1$, $p = \text{Swap}(v, 4) = 3421$. The corresponding three ISTs rooted at vertex $\mathbf{1}_4$ for B_4 are shown in Fig. 2.

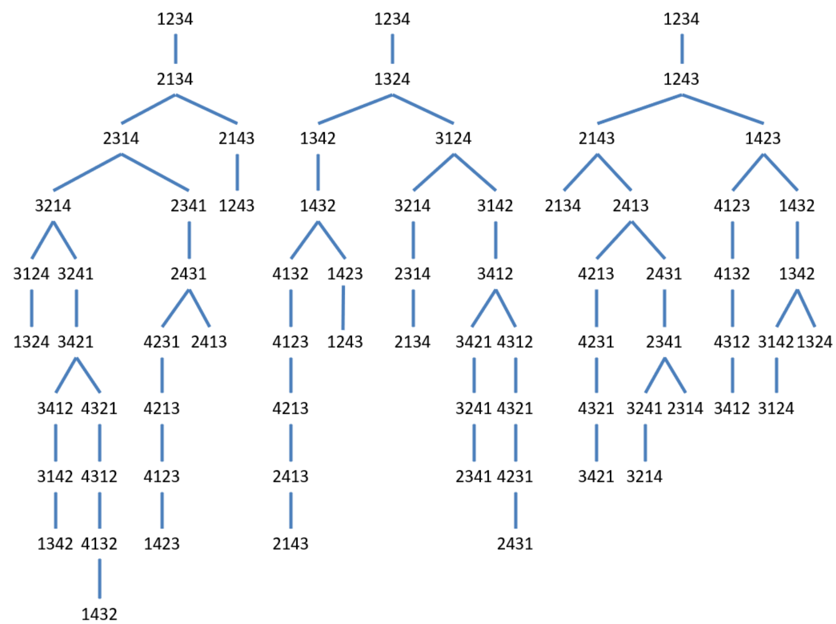

Fig. 2. The three ISTs of B_4 calculated by Algorithm 1

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_t^n and $T_{t'}^n$ for $t, t' \in \{1, 2, \ldots, n-1\}$ with $t \neq t'$, we denote by $T_t^n(v, r)$ and $T_{t'}^n(v, r)$ the two paths from v to the common r.

Theorem 1. *For $n \geqslant 4$, $T_1^n, T_2^n, \ldots, T_{n-1}^n$ are $n-1$ ISTs of B_n.*

Proof. Suppose that $n \geqslant 4$, let $r = \mathbf{1}_n (= 12 \cdots n)$, 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 $T_1^n, T_2^n, \ldots, T_{n-1}^n$ 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 $T_t^n(v, r)$ and $T_{t'}^n(v, r)$ (apart from $T_{n-1}^n(v, r)$) 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, $T_t^n(v, r)$ and $T_{t'}^n(v, r)$ are vertex-disjoint. Now consider $T_{n-1}^n(v, r)$, 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, $T_t^n(v, r)$, $T_{t'}^n(v, r)$ and $T_{n-1}^n(v, r)$ are vertex-disjoint. See Fig. 3, the paths from the vertex v with $v_n = n$ to r are marked in red, in $T_{n-1}^n(v, r)$ each vertex of the path has symbol $n-1$ in v_n. The other trees $T_t^n(v, r)$ have symbol t in position v_n for $t \in \{1, 2, \ldots, n-2\}$.

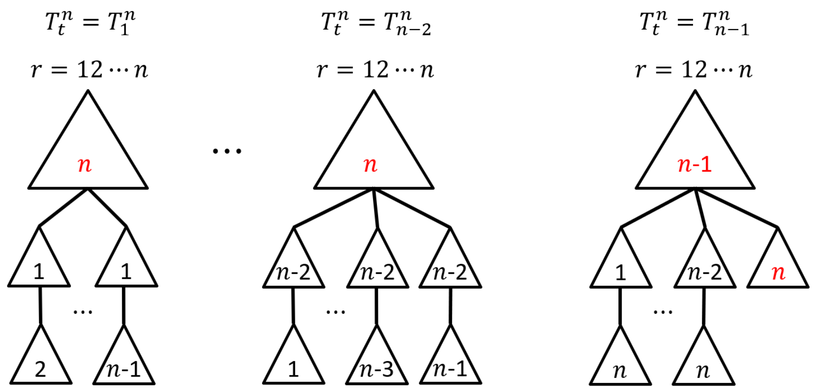

Fig. 3. An illustration of the paths described in the proof of Case 1 of Theorem 1

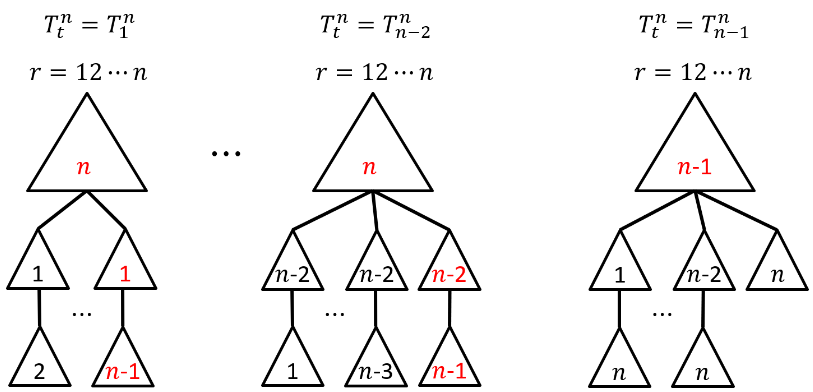

Fig. 4. An illustration of the paths described in the proof of Case 2 of Theorem 1

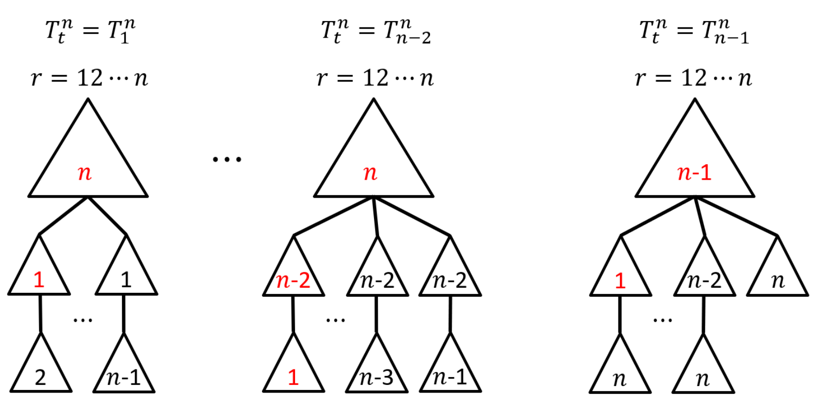

Fig. 5. An illustration of the paths described in the proof of Case 3 of Theorem 1

Case 2: $v_n = n-1$.

Each vertex of the two paths $T_t^n(v,r)$ and $T_{t'}^n(v,r)$ (apart from $T_{n-1}^n(v,r)$) 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, $T_t^n(v,r)$ and $T_{t'}^n(v,r)$ are vertex-disjoint. In $T_{n-1}^n(v,r)$ each vertex of the path swaps symbol n to the position v_n. By Case 1, the paths $T_1^n(v,r)$ and $T_{n-2}^n(v,r)$ are vertex-disjoint. Hence, $T_t^n(v,r)$, $T_{t'}^n(v,r)$ and $T_{n-1}^n(v,r)$ are vertex-disjoint. See Fig. 4, the paths from the vertex v with $v_n = n-1$ to r are marked in red, in $T_1^n(v,r)$ each vertex of the path has symbol $n-1$, 1 or n in the position v_n, in $T_{n-2}^n(v,r)$ each vertex of the path has symbol $n-1$, $n-2$ or n in the position v_n, in $T_{n-1}^n(v,r)$ 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 $T_t^n(v,r)$ and $T_{t'}^n(v,r)$ (apart from $T_{n-1}^n(v,r)$) 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, $T_t^n(v,r)$ and $T_{t'}^n(v,r)$ are vertex-disjoint. In $T_{n-1}^n(v,r)$ each vertex of the path swaps symbol $n-1$ to v_n. By Case 2, the paths $T_1^n(v,r)$ and $T_{n-2}^n(v,r)$ are vertex-disjoint. Hence, $T_t^n(v,r)$, $T_{t'}^n(v,r)$ and $T_{n-1}^n(v,r)$ are vertex-disjoint. See Fig. 5, the paths from the vertex v with $v_n = 1$ to r are marked in red, in $T_1^n(v,r)$ each vertex of the path swaps symbol n to v_n, in $T_{n-2}^n(v,r)$ each vertex of the path swaps symbol $n-2$ or n to v_n, in $T_{n-1}^n(v,r)$ 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 $H_n = \max_{1 \leqslant t \leqslant n-1} h(T_t^n)$ 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 $\mathcal{O}(n \cdot n!)$ 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 $\mathcal{O}(n \cdot n!)$. Hence, the total time complexity $\mathcal{O}(n \cdot n!)$ 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 $\mathcal{O}(n \cdot n!)$ of our algorithm is asymptotically optimal, where n is the dimension of B_n and $n!$ is the number of vertices of the network.

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

1. S. B. Akers and B. Krishnamurty, "A group theoretic model for symmetric interconnection networks," *IEEE Transactions on Computers*, vol. 38, no. 4, pp. 555-566, 1989.
2. F. Bao, Y. Funyu, Y. Hamada, and Y. Igarashi, "Reliable broadcasting and secure distributing in channel networks," in: *Proc. of 3rd International Symposium on Parallel Architectures, Algorithms and Networks*, ISPAN'97, Taipei, December 1997, pp. 472-478.
3. J.-M. Chang, J.-D. Wang, J.-S. Yang, and K.-J. Pai, "A comment on independent spanning trees in crossed cubes," *Information Processing Letters*, vol. 114, no. 12, pp. 734-739, 2014.
4. J.-M. Chang, T.-J. Yang, and J.-S. Yang, "A parallel algorithm for constructing independent spanning trees in twisted cubes," *Discrete Applied Mathematics*, vol. 219, pp. 74-82, 2017.
5. Y.-H. Chang, J.-S. Yang, J.-M. Chang, and Y.-L. Wang, "A fast parallel algorithm for constructing independent spanning trees on parity cubes," *Applied Mathematics and Computation*, vol. 268, pp. 489-495, 2015.
6. Y.-H. Chang, J.-S. Yang, S.-Y. Hsieh, J.-M. Chang, and Y.-L. Wang, "Construction independent spanning trees on locally twisted cubes in parallel," *Journal of Combinatorial Optimization*, vol. 33, no. 3, pp. 956-967, 2017.

7. D.-W. Cheng, C.-T. Chan, and S.-Y. Hsieh, "Constructing independent spanning trees on pancake networks," *IEEE Access*, vol. 8, pp. 3427-3433, 2020.
8. D.-W. Cheng, K.-H. Yao, and S.-Y. Hsieh, "Constructing Independent Spanning Trees on Generalized Recursive Circulant Graphs," *IEEE Access*, vol. 9, pp. 74028-74037, 2021.
9. J. Cheriyan and S.N. Maheshwari, "Finding nonseparating induced cycles and independent spanning trees in 3-connected graphs," *Journal of Algorithms*, vol. 9, no. 4, pp. 507-537, 1988.
10. S. Curran, O. Lee, and X. Yu, "Finding four independent trees," *SIAM Journal on Computing*, vol. 35, no. 5, pp. 1023-1058, 2006.
11. T. Hasunuma, "Completely independent spanning trees in maximal planar graphs," in: *Proc. of 28th International Workshop on Graph-Theoretic Concepts in Computer Science*, WG 2002, LNCS, vol. 2573, Springer, Cham, 2002, pp. 235-245.
12. J.-F. Huang, E. Cheng, and S.-Y. Hsieh, "Two algorithms for constructing independent spanning trees in (n, k)-star graphs," *IEEE Access*, vol. 8, pp. 175932-175947, 2020.
13. J.-F. Huang, S.-S. Kao, S.-Y. Hsieh, and R. Klasing, "Top-Down construction of independent spanning trees in alternating group networks," *IEEE Access*, vol. 8, pp. 112333-112347, 2020.
14. A. Itai and M. Rodeh, "The multi-tree approach to reliability in distributed networks," *Information and Computation*, vol. 79, no. 1, pp. 43-59, 1988.
15. S.-S. Kao, K.-J. Pai, S.-Y. Hsieh, R.-Y. Wu, and J.-M. Chang, "Amortized efficiency of constructing multiple independent spanning trees on bubble-sort networks," *Journal of Combinatorial Optimization*, vol. 38, no. 3, pp. 972-986, 2019.
16. S. Lakshmivarahan, J. Jwo, and S. K. Dhall, "Symmetry in interconnection networks based on Cayley graphs of permutation groups: A survey," *Parallel Computing*, vol. 19, no. 4, pp. 361-407, 1993.
17. F.T. Leighton, "Introduction to Parallel Algorithms and Architectures: Arrays, Trees, Hypercubes," *Morgan Kaufmann Publishers*, 1992.
18. L.-H. Liu, J.-E. Chen, S.-Q. Chen, and W.-J. Jia, "An new representation for interconnection network structures," *Journal of Central South University of Technology*, vol. 9, no. 1, pp. 47-53, 2002.
19. C.-F. Lin, J.-F. Huang, and S.-Y. Hsieh, "Constructing independent spanning trees on transposition networks," *IEEE Access*, vol. 8, pp. 147122-147132, 2020.
20. J.-C. Lin, J.-S. Yang, C.-C. Hsu, and J.-M. Chang, "Independent spanning trees vs. edge-disjoint spanning trees in locally twisted cubes," *Information Processing Letters*, vol. 110, no. 10, pp. 414-419, 2010.
21. K.-J. Pai, J.-S. Yang, S.-C. Yao, S.-M. Tang, and J.-M. Chang, "Completely independent spanning trees on some interconnection networks," *IEICE Transactions on Information Systems*, vol. E97-D, no. 9, pp. 2514-2517, 2014.
22. K.-J. Pai and J.-M. Chang, "Dual-CISTs: Configuring a protection routing on some Cayley networks," *IEEE/ACM Transactions on Networking*, vol. 27, no. 3, pp. 1112-1123, 2019.
23. K.-J. Pai, R.-S. Chang, R.-Y. Wu, and J.-M. Chang, "A two-stages tree-searching algorithm for finding three completely independent spanning trees," *Theoretical Computer Science*, vol. 784, pp. 65-74, 2019.
24. K.-J. Pai, R.-S. Chang, Ro-Yu Wu, and J.-M. Chang, "Three completely independent spanning trees of crossed cubes with application to secure-protection routing," *Information Sciences*, vol. 541, pp. 516-530, 2020.

25. K.-J. Pai, R.-S. Chang, and J.-M. Chang, "Constructing dual-CISTs of pancake graphs and performance assessment of protection routings on some Cayley networks," *The Journal of Supercomputing*, http://dx.doi.org/10.1007/s11227-020-03297-9
26. F. Péterfalvi, "Two counterexamples on completely independent spanning trees," *Discrete Mathematics*, vol. 312, no. 4, pp. 808-810, 2012.
27. N. Shawash, "Relationships among popular interconnection networks and their common generalization," *Ph.D. thesis, Oakland University*, 2008.
28. S.-M. Tang, J.-S. Yang, Y.-L. Wang, and J.-M. Chang, "Independent spanning trees on multidimensional torus networks," *IEEE Transactions on Computers*, vol. 59, no. 1, pp. 93-102, 2010.
29. Y. Wang, J. Fan, G. Zhou, and X. Jia, "Independent spanning trees on twisted cubes," *Journal of Parallel and Distributed Computing* vol. 72, no. 1, pp. 58-69, 2012.
30. J.-S. Yang, H.-C. Chan, and J.-M. Chang, "Broadcasting secure messages via optimal independent spanning trees in folded hypercubes," *Discrete Applied Mathematics*, vol. 159, no. 12, pp. 1254-1263, 2011.
31. J.-S. Yang and J.-M. Chang, "Optimal independent spanning trees on Cartesian product of hybrid graphs," *Computer Journal*, vol. 57, no. 1, pp. 93-99, 2014.
32. J.-S. Yang, J.-M. Chang, K.-J. Pai, and H.-C. Chan, "Parallel construction of independent spanning trees on enhanced hypercubes," *IEEE Transactions on Parallel and Distributed Systems*, vol. 26, no. 11, pp. 3090-3098, 2015.
33. J.-S. Yang, J.-M. Chang, S.-M. Tang, and Y.-L. Wang, "Reducing the height of independent spanning trees in chordal rings," *IEEE Transactions on Parallel and Distributed Systems*, vol. 18, no. 5, pp. 644-657, 2007.
34. J.-S. Yang, J.-M. Chang, S.-M. Tang, and Y.-L. Wang, "On the independent spanning trees of recursive circulant graphs $G(cd^m, d)$ with $d > 2$," *Theoretical Computer Science*, vol. 410, no. 21-23, pp. 2001-2010, 2009.
35. J.-S. Yang, J.-M. Chang, S.-M. Tang, and Y.-L. Wang, "Constructing multiple independent spanning trees on recursive circulant graphs $G(2^m, 2)$," *International Journal of Foundations of Computer Science*, vol. 21, no. 1, pp. 73-90, 2010.
36. J.-S. Yang, S.-S. Luo, and J.-M. Chang, "Pruning longer branches of independent spanning trees on folded hyper-stars," *Computer Journal*, vol. 58, no. 11, pp. 2972-2981, 2015.
37. J.-S. Yang, S.-M. Tang, J.-M. Chang, and Y.-L. Wang, "Parallel construction of optimal independent spanning trees on hypercubes," *Parallel Computing*, vol. 33, no. 1, pp. 73-79, 2007.
38. J.-S. Yang, M.-R. Wu, J.-M. Chang, and Y.-H. Chang, "A fully parallelized scheme of constructing independent spanning trees on Möbius cubes," *Journal of Supercomputing*, vol. 71, no. 1, pp. 894-908, 2015.
39. Y.-C. Yang, S.-S. Kao, R. Klasing, S.-Y. Hsieh, H.-H. Chou, and J.-M. Chang, "The construction of multiple independent spanning trees on burnt pancake networks," *IEEE Access*, vol. 9, pp. 16679-16691, 2021.
40. A. Zehavi and A. Itai, "Three tree-paths," *Journal of Graph Theory*, vol. 13, no. 2, pp. 175-188, 1989.
41. S.-L. Zhao and R.-X Hao, "The generalized connectivity of (n, k)-bubble-sort graphs," *The Computer Journal*, vol. 62, no. 9, pp. 1277-1283, 2019.
42. S.-L. Zhao and R.-X Hao, "The fault tolerance of (n, k)-bubble-sort networks," *Discrete Applied Mathematics*, vol. 285, pp. 204-211, 2020.