Two Algorithms for Constructing Independent Spanning Trees in \((n,k)\)-Star Graphs

JIE-FU HUANG\textsuperscript{1}, EDDIE CHENG\textsuperscript{2}, AND SUN-YUAN HSIEH\textsuperscript{3}, (Senior Member, IEEE)

\textsuperscript{1}Department of Computer Science and Information Engineering, National Cheng Kung University, Tainan 701, Taiwan
\textsuperscript{2}Department of Mathematics and Statistics, Oakland University, MI 48309, USA
\textsuperscript{3}Department of Computer Science and Information Engineering, Institute of Medical Informatics, Institute of Manufacturing Information and Systems, International Center for the Scientific Development of Shrimp Aquaculture, Center for Innovative FinTech Business Models, National Cheng Kung University, Tainan 701, Taiwan

Corresponding author: Sun-Yuan Hsieh (hsiehsy@mail.ncku.edu.tw)

\textbf{ABSTRACT} In a graph \(G\), two spanning trees \(T_1\) and \(T_2\) are rooted at the same vertex \(r\). If, for every \(v \in V(G)\), the paths from \(v\) to the root \(r\) in \(T_1\) and \(T_2\) are internally vertex-disjoint, they are called independent spanning trees (ISTS). ISTs can be applied in numerous fields, such as fault-tolerant broadcasting and secure message distribution. The \((n,k)\)-star graphs \(S_{n,k}\) constitute a generalized version of the star network. The method of constructing ISTs in \((n,k)\)-star graphs remains unknown. In this paper, we propose one recursive algorithm and one parallel algorithm for constructing ISTs on \((n,k)\)-star graphs. The main ideas of the recursive algorithm are to use induction to change small trees into large trees, to use a modified breadth-first search (MBFS) traversal to create a frame for an IST, and to use a breadth-first search (BFS) traversal to connect the rest of nodes. The main ideas of the parallel algorithm are to create frames through MBFS traversals in parallel, and to use some specific rules to connect the rest of nodes in parallel. We also present validation proofs for both algorithms, and analyze the time complexities of both algorithms. The time complexity of the recursive algorithm in \(S_{n,k}\) is \(O(n \times \frac{n!}{(n-k)!})\), where \(\frac{n!}{(n-k)!}\) is the number of nodes of \(S_{n,k}\). The time complexity of the parallel algorithm can be reduced to \(O(\frac{n!}{(n-k)!})\) if the system has \(n - 1\) processors computing in parallel. Both algorithms are correct with the stated time complexity values; the parallel algorithm is more efficient than the recursive algorithm.

\textbf{INDEX TERMS} Independent spanning trees, \((n,k)\)-star graphs, breadth-first search, recursive algorithm, parallel algorithm.

I. INTRODUCTION

In a graph \(G\), two spanning trees \(T_1\) and \(T_2\) are rooted at the same vertex \(r\). If, for every \(v \in V(G)\), the paths from \(v\) to the root \(r\) in \(T_1\) and \(T_2\) are internally vertex-disjoint, they are called independent spanning trees (ISTS). ISTs can be applied in numerous fields, such as fault-tolerant broadcasting and secure message distribution [1], [22]. Assume that a network \(N\) has \(k\) ISTs rooted at node \(r\), and \(N\) contains at most \(k - 1\) faulty nodes. These applications are illustrated as follows [17]:

- In fault-tolerant broadcasting, node \(r\) broadcasts a message to every non-faulty node \(v\) in \(N\) through the \(k\) ISTs. Because the number of faulty nodes is less than \(k\), at least one of the \(k\) internally disjoint paths from \(r\) to \(v\) is fault free; this promises the message can be delivered to every node in \(N\) reliably;

- In secure message distribution, node \(r\) divides a message into \(k\) packets, and sends each packet to the destination through a different IST. Thus, each node in \(N\) receives at most one of the \(k\) packets except the destination node, which receives all \(k\) packets.

Scholars have described how to construct ISTs in graphs. Zehavi and Itai proposed a conjecture that there exist \(k\) ISTs in a \(k\)-connected graph [28]. The conjecture has been proven correct for \(k\)-connected graphs such that \(k \leq 4\) [9], [12], [18], but it is still unknown for graphs such that \(k \geq 5\). However, the problem is considerably challenging for arbitrary graphs, and from 2007 to 2020, researchers have published studies on specifying ISTs in interconnection networks.

The constructions of ISTs on several interconnection networks have been solved, including crossed cubes [7], Möbius cubes [8], twisted cubes [24], locally twisted cubes [13], [21],
hypervolumes [26], parity cubes [23], folded hypercubes [27], and bubble-sort networks [19], [20], alternating group networks [16]. In those studies, some parallel algorithms have been proposed on twisted cubes [5], locally twisted cubes [4], parity cubes [3], and hypercubes [25], [26]. To the best of our knowledge, the method of constructing ISTs in \((n,k)\)-star graphs is still unknown. In this paper, we concentrate on \((n,k)\)-star graphs.

An \((n,k)\)-star graph \(S_{n,k}\) refers to a generalized version of an \(n\)-star graph \(S_n\), where \(S_{n,k-1}\) and \(S_n\) are isomorphic and \(S_{n,1}\) is obviously a complete graph \(K_n\). Scholars have computed or derived some basic properties of graphs of the form \(S_{n,k}\), such as diameter [10], connectivity [10], broadcasting [6], average distance [11], embedding [2], Hamiltonicity [14], spanning connectivity [15], and wide diameter [15]. These results demonstrate that \(S_{n,k}\) graphs have excellent topological properties.

In this paper, we propose one recursive algorithm and one parallel algorithm. Both of which construct ISTs in \((n,k)\)-star graphs. The main ideas of the recursive algorithm are to use induction to extend small trees into big trees, use a modified breadth-first search (MBFS) traversal to create a frame of an IST, and use a breadth-first search (BFS) traversal to connect the remaining nodes. The main ideas of the parallel algorithm are to create frames through MBFS traversals in parallel and use the specific rules to connect the rest of nodes in parallel.

In this paper, we validate both algorithms, and analyze the time complexity of both algorithms. The time complexity of the recursive algorithm in \(S_{n,k}\) is \(O(n \times \frac{nk}{(n-k)2})\), where \(\frac{nk}{(n-k)2}\) is the number of nodes of \(S_{n,k}\), whereas that of the parallel algorithm can decline to \(O(\frac{nk}{(n-k)2})\) if the system has \(n-1\) processors computing in parallel. Both algorithms are correct with the stated time complexity; furthermore, the parallel algorithm is more efficient than the recursive algorithm.

The remainder of this paper is organized as follows: Section II explains preliminary considerations, Section III presents both algorithms, Section IV proves the relevant claims, Section V analyzes and compares both algorithms, and Section VI concludes the paper.

II. PRELIMINARY CONSIDERATIONS

Let \(n = \{1, 2, \ldots, n\}\) and \(k = \{1, 2, \ldots, k\}\).

Definition 1 ([10]): An \((n,k)\)-star graph, denoted by \(S_{n,k}\), is specified by two integers \(n\) and \(k\), where \(1 \leq k < n\). The node set of \(S_{n,k}\) is denoted by \(\{p_1p_2 \ldots p_k \mid p_i \in (n) \text{ and } p_i \neq p_j \text{ for } i \neq j\}\). The adjacency is defined as follows: \(p_1p_2 \ldots p_i \ldots p_k\) is adjacent to

1. \(p_1p_2 \ldots p_i \ldots p_k\) through an edge of dimension \(i\), where \(2 \leq i \leq k\) (swap \(p_1\) and \(p_i\)) and
2. \(xp_2 \ldots p_i\) through dimension 1, where \(x \in (n) - \{p_i \mid 1 \leq i \leq k\}\).

The node of type (1) is referred to as friend of node \(i\) of \(p_1p_2 \ldots p_i \ldots p_k\), and the node of type (2) is referred to as child of \(p_1p_2 \ldots p_i \ldots p_k\), where \(z\) is the ordinal number of \(x\) in \(\langle n \rangle - \{p_i \mid 1 \leq i \leq k\}\) in ascending order.

\(S_{3,1}\) is depicted in Figure 1. Node 3 has two children, namely node 1 (child 1) and node 2 (child 2). \(S_{4,2}\) is displayed in Figure 2. Node 34 has two children and one friend as presented in Figure 3. \(S_{5,3}\) consists of 60 nodes and 120 edges, as displayed in Figure 4, and is composed of five instances \(S_{4,2}\).

![Figure 1](image1.png)

**FIGURE 1.** \(S_{3,1}\).

![Figure 2](image2.png)

(a) Child 1. (b) Child 2. (c) Friend 2.

**FIGURE 2.** \(S_{4,2}\).

![Figure 3](image3.png)

**FIGURE 3.** Children and friend of node 34 in \(S_{4,2}\).

![Figure 4](image4.png)

**FIGURE 4.** \(S_{5,3}\).

Like the \(n\)-star graph \(S_n\), an \(S_{n,k}\) can be decomposed into \(n\) instances of \(S_{n-1,k-1}\). That is, an \(S_{n,k}\) can be decomposed into \(n\) vertex-disjoint instances of \(S_{n-1,k-1}\) in \(k - 1\) different ways by fixing the symbol in any position \(i\), such
that $2 \leq i \leq k$. This decomposition can be conducted recursively on each $S_{n-1,k-1}$ to obtain smaller subgraphs.

**Lemma 1 ([10]):** $(n, k)$-star graphs is vertex symmetric. According to Lemma 1, any node in $S_{n,k}$ can be the root in the process of constructing ISTs.

**Lemma 2 ([10]):** $(n, n-1)$-star graph $S_{n,n-1}$ is isomorphic to the $n$-star graph $S_n$

### III. ALGORITHMS

Notations used here are defined as follows:

- the root: node $(n-k+1)(n-k+2)(n-k+3)\ldots n$;
- cluster A: a set of nodes whose last symbol is $A$;
- the root cluster: the set of nodes whose last symbol is identical to that of the root. In $S_{n,k}$, the root cluster is cluster $n$;
- frame: a set of nodes with the same last symbols created by buildFrame function, which is defined later in a later passage;
- germ: the starting node of a frame;
- $T_{j}\bar{n}^{k}$: the $j$th IST of $S_{n,k}$, and the last symbol of its frame is $j$;
- tid: the last symbol of the germ, namely $j$ of $T_{j}\bar{n}^{k}$.

$S_{n,k}$ is partitioned into $n$ instances of $S_{n-1,k-1}$. Every node is classified into a cluster by its last symbol. Thus, $S_{n,k}$ has $n$ clusters. For example, $S_{4,2}$ can be divided into four instances of $S_{3,1}$. $S_{4,2}$ has four clusters, as illustrated in Figure 5.

![Four clusters of $S_{4,2}$](image)

![Germs $r_k = n - k + i$; hence, $r_k = n$.](image)

Child 1 of the root in $S_{n,k}$ uses its new edge to make the germ of $T_{n}\bar{n}^{k}$. Child 2 of the root in $S_{n,k}$ uses its new edges to make the germ of $T_{n-1}^{n}$. The root in $S_{n,k}$ uses its new edge to make the germ of $T_{n-1}^{n},$ namely friend $k$. Friends 3 to $k - 1$ of the root in $S_{n,k}$ use their new edges to make the germs of $T_{n-1}^{n-k+3}$ to $T_{n-1}^{n-k+2}$. $T_{n-1}^{n-k+2}$ does not conduct any buildFrame function; consequently, it has no germ, and that is why we skip friend 2 in Figure 6.

**Basic variable and functions:**

- tary: an array of ISTs of this iteration; it is a two-dimensional array, and $\text{tary}[$tid$][\text{node}]$ stores its parent. Therefore, every IST includes all nodes. If the node has not yet been traversed, $\text{tary}[$tid$][\text{node}] = -1$.
- $\text{ptary}$: an array of the trees of the previous iteration.
- $\text{edge}$: an associative array for storing all direct edges only used in the recursive algorithm. Initially, $\text{edge}[$from$][to] = 1$ means an unused direct edge; a $\text{edge}[$from$][to] = 0$ means a used direct edge.
- function getChildren(parent): returns the children of parent.
- function getFriend(parent, $p$): returns the friend $p$ of parent by swapping the first and the $p$th symbols of parent.
- function checkChild(tary, tid, edge, parent, children): returns true if any child of parent in children is unvisited in Tree tid, false otherwise. $\text{edge}$ is not required and set to false in the parallel algorithm.

**A. buildFrame FUNCTION**

A frame tree can be constructed from a germ through a modified BFS (MBFS) traversal. However, the MBFS traversal must stop when it encounters a node of which the last symbol differs from that of the germ.

**BFS order.** In the buildFrame function, each node in $fr\text{Que}$ must traverse all unvisited nodes connected to it in this order:
1. child 1, 2, 3, ..., $n - k$;
2. friend 2, 3, 4, ..., $k - 1$. 

![Diagram](image)
At the start of every iteration, the program should copy previous trees so that correct information can be used in the next iteration.

In such case, we should transfer this idea.

MBFS traversal. Because \( S_{n,k} \) is symmetric, some child \( c \) of one node \( v \) may be traversed by another node \( w \) earlier. In such case, we should transfer \( c \)'s parent from \( w \) to \( v \). For instance, in \( S_{n,n-2} \), the graph presented in Figure 7(a) occurs. We set the edge (marked with a blue X) from \( w \) to \( v \), hence, the triangle shape is retained. Figure 7(b) illustrates this idea.

**FIGURE 7.** BFS and MBFS traversals.

For example, in Figures 8, 9, and 10, the frame nodes of \( T_{1}^{5,3}, T_{2}^{5,3}, \) and \( T_{3}^{5,3} \) are indicated by ellipses with blue borders.

**B. BuildLeaf FUNCTION**

The buildLeaf function connects all unvisited nodes one step at a time from a tree after the buildFrame function. For example, in Figure 11, the leaf nodes of \( T_{4}^{5,3} \) in the first, second, third, and fourth buildLeaf executions are indicated by ellipses with brown, gold, gray, and red borders, respectively.

**C. RECURSIVE ALGORITHM**

The recursive algorithm is presented in Algorithm 1.

\( S_{n-k+1,1} \), the basic part, is a complete graph, having \( n - k \) independent spanning trees that are illustrated in Figure 12.

**Copy previous trees.** At the end of each iteration, the trees that had been used must be stored in a tree array called ptary so that correct information can be used in the next iteration. At the start of every iteration, the program should copy previous trees to the tree array tary. Table 1 presents the mapping.

**TABLE 1.** Tree id and previous tree id mapping.

```markdown
| Tree id | 1   | 2   | ... | n-2 | n-1 |
|---------|-----|-----|-----|-----|-----|
| Previous tree id | 1   | 2   | ... | n-2 | n-1 |
```

In a typical iteration, \( T_{1}^{n,k} \) copies information from \( T_{1}^{n-1,k-1} \) and \( T_{n-1,k}^{n,k} \) copies information from \( T_{n-k+1}^{n,k} \). \( T_{n-k+1}^{n,k} \) is a new tree, and thus no previous tree is copied.

Create a framelist. A framelist is an array for storing the germs. The germs are created from the root. In \( S_{4,2} \), we can create three germs, namely 41, 42, and 43 to establish frames, as illustrated in Figure 13.

For example, \( S_{5,3} \) has four trees and three germs: 541 in \( T_{1}^{5,3} \), 542 in \( T_{2}^{5,3} \), and 543 in \( T_{3}^{5,3} \), as presented Figures 8, 9, and 10, respectively. As indicated in Figure 11, \( T_{4}^{5,3} \) does not execute any buildFrame function, but it does conduct buildLeaf functions.
D. BuildFrameP FUNCTION

The frame of an IST comprises nodes of the same tail portion within this IST. The frame of an IST can be constructed from a germ through a modified breadth-first search (MBFS) traversal displayed in Figure 7. However, the MBFS traversal must stop when it encounters a node of which the last tail symbol(s) differ(s) from that of the germ.

**BFS order.** In the buildFrameP function, every node in frQue queue must visit all hitherto unvisited nodes connected to it in the following order:
E. FindParent Function

Let $Frame_{j}^{n,k}$ denote that the frame of $T_{j}^{n,k}$. In $T_{j}^{n,k}$ construction, each unvisited node child whose last symbol is $z$ should be connected to $Frame_{j}^{n,k}$ after the buildFrameP functions. The findParent function is used to find the parent of child in $T_{j}^{n,k}$ by means of the findTarget function according to the following rules.

- **P1 rule:**
  If child’s first symbol is $j$, the findTarget function swaps the first and last symbols of child to return the target node;
- **P2 rule:**
  If child’s first symbol is not $j$, the findTarget function moves $j$ to first position of child by child, or friend operation to return the target node;
- **P3 rule:**
  If the target node has been used by $Frame_{j}^{n,k}$, the target is changed to $n - k + 2$, and the findTarget function is used again to move $n - k + 2$ to the first or last position of child to return another target node;
- **$T_{n-k+1}$ rule:**
  If child is in the root cluster and $j$ is $n - k + 1$, the findParent function swaps the first and last symbols of child to form its parent.
Function buildLeaf(tary, tid, dedge, first, n, k)

Input : tary, tid, dedge, first, n, k
Output: nodes can be visited by tary[tid] in a step

// global means using global variable
1 global lfQue;  // a two-dimensional array storing
2 // nodes visited by Ttid not to visit all nodes next time
3 global lfQIx;  // the starting position in lfQue[tid]
4
5 if first == true then
6  // first execution
7  for every node v in tary[tid] do
8    // only Tn−k+2 can grow from the root cluster
9      if v has parent and ( (tid ≠ n − k + 2 and the
10        last symbol of v ≠ the last symbol of the root) or
11          tid == n − k + 2) then
12        children = getChild(v);
13        for each element c in children do
14          // visit child c
15          if tary[tid][c] == 0 and
16            dedge[v][c] == 1 then
17            tary[tid][c] = v; dedge[v][c] = 0;
18            lfQue[tid][c] = v;  // (1)(2)(3)
19      // visit the friend 2 to k
20      for i = 2; i ≤ k; i = i + 1 do
21        y = lfQIx[tid]; y < qs; y = y + 1 do
22          v = lfQue[tid][y];
23          children = getChild(v);
24          for each element c in children do
25            // visit child c
26            if tary[tid][c] == 0 and dedge[v][c] == 1 then
27              tary[tid][c] = v; dedge[v][c] = 0;
28              lfQue[tid][c] = v;  // (1)(2)(3)
29      // visit the friend 2 to k
30      for i = 2; i ≤ k; i = i + 1 do
31        y = lfQIx[tid]; y < qs; y = y + 1 do
32          v = lfQue[tid][y];
33          children = getChild(v);
34          for each element c in children do
35            // visit child c
36            if tary[tid][c] == 0 and dedge[v][c] == 1 then
37              tary[tid][c] = v; dedge[v][c] = 0;
38              lfQue[tid][c] = v;  // (1)(2)(3)
39    // visit the friend 2 to k
40    for i = 2; i ≤ k; i = i + 1 do
41      y = lfQIx[tid]; y < qs; y = y + 1 do
42        v = lfQue[tid][y];
43        children = getChild(v);
44        for each element c in children do
45          // visit child c
46          if tary[tid][c] == 0 and dedge[v][c] == 1 then
47            tary[tid][c] = v; dedge[v][c] = 0;
48            lfQue[tid][c] = v;  // (1)(2)(3)
49          // (1)Set its parent. (2)Set the edge used (from 1 to 0).
50          // (3)Put it into lfQue[tid].
Function buildFrame(\(tary, tid, germ, tailN\))

Input: \(tary, tid, germ, tailN\)
Output: the frame of \(tary[tid]\) whose last \(tailN\) symbol(s) is(are) fixed

1. \(frQue=\text{array}(\text{germ});\)  
\(>\) Put germ into \(frQue\)

2. 
\(>\) initially.

3. for \(j = 0; j < [frQue]; j = j + 1\) do

4. \(v = frQue[j];\)  
\(>\) the \((j + 1)\)th element of \(frQue\).

5. \(\text{children} = get\text{Child}(v);\)

6. // any child unvisited

7. if checkChild(\(tary, tid, false, v, \text{children}\)) then

8. for each element \(c \) in children do

9. if \(c\) has been visited then

10. remove \(c\) from \(frQue\);

11. // visit child \(c\)

12. \(\text{tary[tid][}c\} = v;\)  
\(>\) Set \(c\)’s parent.

13. \(frQue[\] = \(c;\)  
\(>\) Put \(c\) into \(frQue\).

14. // visit the friend 2 to \(|\text{germ}|-tailN\)

15. for \(i = 2; i \leq \text{the number of symbols of} \)

16. \(\text{germ} - tailN; i = i + 1\) do

17. \(fd = get\text{Friend}(v, i);\)

18. if \(\text{tary[tid][}\] \(= 0\) then

19. \(\text{tary[tid][}fd\} = v;\)  
\(>\) Set \(fd\)’s parent.

20. \(frQue[\] = \(fd;\)  
\(>\) Put \(fd\) into \(frQue\).

21. // Red text indicates a portion that differs from the

22. // buildFrame function.

Function findParent(\(tary, tid, child, diff\))

Input: \(tary, tid, child, diff\)

Output: \(child\)’s parent in \(tary[tid]\)

1. \(\text{len} = \text{the number of} \text{child’s symbols;} \)  
\(>\) \(\text{len}\) is the

2. \(\text{original dimension} \text{k of} S_{n,k}\).

3. \(\text{dim} = 0;\)  
\(>\) \(\text{dim}\) is the reduced dimension \(k\) of \(S_{n,k}\).

4. \(ls = \text{empty string;} \)  
\(>\) the last symbol

5. // reduce the dimensions of \(S_{n,k}\)

6. for \(\text{dim} = \text{len}; \text{dim} \geq 2; \text{dim} = \text{dim} - 1\) do

7. \(ls = \text{child’s \(\text{dim}^{th}\) symbol;} \)

8. if \(ls \neq \text{dim} + \text{diff} \) then

9. break;  
\(>\) The dimension has been confirmed.

10. else if \(\text{tid} > \text{diff} + 1 \) and \(\text{tid} = \text{dim} + \text{diff} - 1\) then

11. \(\text{tid}=\text{diff}+1;\)  
\(>\) Because \(T_{n,k+1}\) is used by

12. \(\text{by} T_{n-k+1}^{x+1,x+1-n+k}.\)

13. else if \(\text{tid} = \text{diff} + 1\) then

14. return getFriend(substring(\(child, 1, \text{dim}\)), \(\text{dim} + 1, \text{len} - \text{dim}\));

\(>\) \(T_{n-k+1}\) rule

15. \(\text{front} = \text{substring(} \text{child, 1, dim);} \)

16. \(\text{back} = \text{substring(} \text{child, dim} + 1, \text{len} - \text{dim};)\)

17. // \(\text{back}\) is the last \(n - \text{dim}\) symbol(s) of \(\text{child}\) which

18. // is(are) identical to that of the root.

19. // \(\text{back}\) may be a empty string if the last symbol of

20. // \(\text{child}\) is not equal to that of the root initially.

21. if \(\text{tid} = \text{diff} + 2\) then

22. \(\text{temp} = \text{findTarget(} \text{front, dim);} \)

23. \(>\) Because \(T_{n-k+2}\)’s frame is the root cluster.

24. else

25. \(\text{temp} = \text{findTarget(} \text{front, tid);} \)

26. if \(\text{ls} = \text{diff} + 2\) then

27. return \(\text{temp} + \text{back};\)  
\(>\) P2 rule.

28. // Because \(T_{n-k+2}\)’s frame is the root cluster,

29. // the target node is not used by \(Frame^{n,k}_{n-k+2}.\)

30. else if \(\text{ls} = \text{diff} + 1 \) and \(\text{dim} < \text{len}\) and

31. \(\text{tary}[\text{dim} + \text{diff}[\text{child}]= \text{temp} + \text{back}\) then

32. return findTarget(front, diff + 2)+back;

33. \(>\) Because \(T_{n-k+1}^{x,x-n+k}\) is used by \(T_{x+1,x+1-n+k}.\)

34. else if \(\text{tary[}\text{ls][child]}= \text{temp} + \text{back}\) then

35. return findTarget(front, diff + 2)+back;

36. \(>\) P3 rule.

37. else

38. \(\text{return temp} + \text{back};\)  
\(>\) P1 or P2 rule.

\(F.\) Parallel Algorithm

The parallel algorithm is presented in Algorithm 2.

Construct the \(n - k\) ISTs of \(S_{n-k+1,1}\) by hand. \(S_{n-k+1,1}\) has \(n - k\) ISTs illustrated in Figure 12.
Algorithm 2 Parallel Algorithm

Input: $n$ and $k$  \hspace{1cm} \triangleright \text{the dimensions of } S_{n,k}$
Output: the ISTs of $S_{n,k}$

1. Construct the $n-k$ ISTs of $S_{n-k+1,1}$ by hand;
2. if \ $n \geq n - k + 2$ then
3. Create a FrameGermSet;  \hspace{1cm} \triangleright \text{See Algorithm 3.}
4. for each germ in the FrameGermSet do in parallel
5. \hspace{1cm} Execute buildFrameP function;
6. for \ $j = 1; j \leq n - 1; j = j + 1$ do in parallel
7. \hspace{2cm} for each node $v$ in $T^{|n,k}$ do in parallel
8. \hspace{3cm} if $v$ has no parent then
9. \hspace{4cm} Find and set $v$'s parent;

Algorithm 3 Create a FrameGermSet

Input: $n$ and $k$  \hspace{1cm} \triangleright \text{the dimensions of } S_{n,k}$
Output: a FrameGermSet

1. $\text{children} = \text{getChild}(\text{root});$
2. for $w = 2; w \leq k; w = w + 1$ do
3. \hspace{1cm} $\text{glist}$ is an array (a map) that associates
4. \hspace{2cm} values (germs) to keys (tid).
5. \hspace{1cm} $\text{temp} = n - k + 1;$
6. \hspace{2cm} if $w + n - k < n$ then
7. \hspace{3cm} $\text{temp} = w + n - k;$  \hspace{1cm} \triangleright \text{Because } T^{n-k+1,w}_{n-k+1,n-k+1} \text{is used by } T^{n-k+1,w}_{n-k+1,n-k+1}.$
8. \hspace{3cm} \ // Put germs into glist[tid] for $T_1$ to $T_{n-k}$.
9. \hspace{2cm} for each child $i$ in children do
10. \hspace{3cm} \hspace{1cm} glist[i] = getFriend( child $i$, $w$);
11. \hspace{3cm} \hspace{1cm} tary[i][glist[i]] = child $i$;
12. \hspace{3cm} \hspace{1cm} Set the parent of $T_{i}^{w+n-k,w}$'s germ.
13. \hspace{3cm} \hspace{1cm} glist[temp] = getFriend( root, $w$);  \hspace{1cm} \triangleright \text{Put the germ}
14. \hspace{3cm} \hspace{1cm} of $T_{\text{temp}}$ into glist[\text{temp}].
15. \hspace{3cm} \hspace{1cm} tary[temp][glist[temp]] = root;
16. \hspace{3cm} \hspace{1cm} \ // Put germs into glist[tid] for $T_{n-k}$ to $T_{n-k+3}$ to $T_{n-k+w-1}$.
17. \hspace{2cm} for $i = 3; i < w - 1; i = i + 1$ do
18. \hspace{3cm} \hspace{1cm} glist[i+n-k] = getFriend(getFriend( root, $i$),$w$);
19. \hspace{3cm} \hspace{1cm} tary[i+n-k][glist[i+n-k]] =
20. \hspace{3cm} \hspace{1cm} \hspace{1cm} getFriend( root, $i$);
21. \hspace{3cm} \hspace{1cm} \hspace{1cm} Set the parent of germs
22. \hspace{3cm} \hspace{1cm} \hspace{1cm} \hspace{1cm} \hspace{1cm} \hspace{1cm} from $T_{n-k+3}$ to $T_{n-k+w-1}$.
23. \hspace{2cm} for each mapping (tid, germ) in glist do
24. \hspace{3cm} \hspace{1cm} Put ( germ, tid, $k - w + 1$ ) into FrameGermSet;
25. \hspace{3cm} \hspace{1cm} $k - w + 1$ is tailN.

G. CREATE A FrameGermSet

The germs are created from the root. In the recursive algorithm to construct the ISTs of $S_{n,k}$, we should construct the ISTs of $S_{n-k+1,1}$ to $S_{n,k}$ sequentially and iteratively. In the iteration of $S_{x,x-n+k}$, the ISTs of $S_{x,x-n+k}$ must be retained in the next iteration to construct ISTs of $S_{x+1,x+1-n+k}$ as presented in Table 2. The meanings of the symbols in Table 2 are as follows:

- Hand: the IST is constructed by hand;
- New: the IST is constructed without copying the previous ISTs;
- Space: No IST exists;
- $T^{|x-1,x-1-n+k}$: in a lattice($S_{x,x-n+k}$, $T_y$), nodes of $T^{|x-1,x-1-n+k}$ are appended a symbol $x$ and becomes part of $T^{|x,x-n+k}$.

In the parallel algorithm, we create all germs initially, and then construct all ISTs in table 2 except for $T^{n-k+1,1}_{n-k+1,1}, T^{n-k+1,3}_{n-k+1,3}, \ldots, T^{n-k+1,1}_{n-k+1,1}$ (constructed by hand), and $T^{n-k+3,2}_{n-k+3,2}, T^{n-k+4,3}_{n-k+4,3}, \ldots, T^{n-k+1,1}_{n-k+1,1}$ (no buildFrame operation required) in parallel. Notably, $n - k + 1$th IST in $S_{x,x-n+k}$ must be part of the last IST of $S_{x+1,x+1-n+k}$. Namely:
TABLE 2. The previous ISTs used in each dimension of $S_{n,k}$.

|       | $T_{n-k+1,1}$ | $T_{n-k+2,2}$ | $T_{n-k+3,3}$ | $T_{n-k+4,4}$ | $T_{n-k+1,1}$ | $T_{n-k+2,2}$ | $T_{n-k+3,3}$ | $T_{n-k+4,4}$ |
|-------|---------------|---------------|---------------|---------------|---------------|---------------|---------------|---------------|
| $S_{n-k+1,1}$ | T_{n-k+1,1}   | T_{n-k+1,1}   | T_{n-k+1,1}   | T_{n-k+1,1}   | T_{n-k+1,1}   | T_{n-k+1,1}   | T_{n-k+1,1}   | T_{n-k+1,1}   |
| $S_{n-k+2,2}$ | T_{n-k+2,2}   | T_{n-k+2,2}   | T_{n-k+2,2}   | T_{n-k+2,2}   | T_{n-k+2,2}   | T_{n-k+2,2}   | T_{n-k+2,2}   | T_{n-k+2,2}   |
| $S_{n-k+3,3}$ | T_{n-k+3,3}   | T_{n-k+3,3}   | T_{n-k+3,3}   | T_{n-k+3,3}   | T_{n-k+3,3}   | T_{n-k+3,3}   | T_{n-k+3,3}   | T_{n-k+3,3}   |
| $S_{n-k+4,4}$ | T_{n-k+4,4}   | T_{n-k+4,4}   | T_{n-k+4,4}   | T_{n-k+4,4}   | T_{n-k+4,4}   | T_{n-k+4,4}   | T_{n-k+4,4}   | T_{n-k+4,4}   |
| $S_{n-k+5,5}$ | T_{n-k+5,5}   | T_{n-k+5,5}   | T_{n-k+5,5}   | T_{n-k+5,5}   | T_{n-k+5,5}   | T_{n-k+5,5}   | T_{n-k+5,5}   | T_{n-k+5,5}   |

$T_{n-k+1,1}$ is used by $T_{n-k+1,1}$. The process is presented in Algorithm 3.

A FrameGermSet stores elements whose data structure as below:

- a germ;
- tid to indicate which IST uses the germ to create a frame;
- a tail number denoted by tailN to indicate the number of last symbols of a node identical to that of the germ, namely the tail portion. If tailN ≥ 2, the last tailN − 1 symbol(s) of the tail portion is(are) identical of that of the root, and the dimension term of $S_{n,k}$ changes from $(n, k)$ to $(n − tailN + 1, k − tailN + 1)$.

In $S_{n,k}$, to establish frames, we can create three germs: 41, 42, and 43. The tail number of each is 1. They are represented as gray nodes in Figure 14. In $S_{n,k}$ has four ISTs and six germs: 415 (tailN is 2) and 541 (tailN is 1) of $T_{1,5}$, 425 (tailN is 2) and 542 (tailN is 1) of $T_{2,5}$, 543 (tailN is 1) of $T_{3,5}$, and 435 (tailN is 2) of $T_{4,5}$ as gray nodes presented in Figures 15, 16, 17, and 18, respectively. Tree $n-k+2,2$ does not execute any buildFrame function in $S_{n-k+3,3}$ to $S_{n,k}$, but $T_{n-k+2,2}$ is used by $T_{n-k+3,3}$. The tail portion is represented as black symbols behind nodes.

H. PATH COMPOSITION

Finally, the outcome paths from the root to other nodes contain the frame portions and the leaf portions, as illustrated in Figure 19. The tree frames have germs, except in the case of $T_{n-k+2,2}$.

Because $T_{n-k+2,2}$ done not execute any buildFrame function, it appends the last symbol of the root to its previous tree as its frame. For instance, the green nodes in $T_{5,3}$ (Figure 11) come from $T_{3,3}$ (Figure 13(c)), and the symbol 5 appended to these nodes to form the frame of $T_{4,3}$.

IV. PROOFS

Lemma 3: In $S_{n,k}$, each node belongs to a complete graph of $n − k + 1$ nodes.

Proof: In $S_{n,k}$, each node has $n − k$ children. Those $n − k + 1$ nodes are adjacent to each other and form a complete graph.

Lemma 4: In the frame portion, if one node has one child, it must have the other $n − k − 1$ children.

Proof: The MBFS traversal is applied in the buildFrame function; hence each parent will have $n − k$ children in the frame.

Lemma 5: In $S_{n,k}$, each node supplies an incoming edge for each tree.

Proof: We know that $S_{n,k}$ is connected and $n − 1$ ISTs exist. Assume that some node in $S_{n,k}$ supplies its two incoming edges to some tree, then this node must be traversed twice by the tree. The situation is not reasonable, and will not occur. This notion is represented in Figure 20, where different colors represent different trees.
Lemma 6: Each tree can traverse all nodes in cluster $n - k + 2$ in at most two steps.

Proof: We know that $T_{n-k+2}$ does not execute any build-Frame function. Thus, all edges in cluster $n - k + 2$ are not used. In cluster $n - k + 2$, each node is linked directly by some tree. If some node in cluster $n - k + 2$ is not directly linked by a tree, it can link its children and friends to reach the tree. Let $X = n - k + 2$. Consider one node $cdX$ in cluster $X$. Tree
The following examples.

1. \( T_i \):
   - **Root**
   - **Frame**
   - **Leaf**
   
   \[ T_{n-k+1} \text{ (Root) } \rightarrow \text{Frame} \rightarrow \text{Leaf} \]
   
2. \( T_{n-k+2} \):
   - **Root**
   - **Frame**
   - **Leaf**
   
   \[ T_{n-k+2} \text{ (Root) } \rightarrow \text{Frame} \rightarrow \text{Leaf} \]

**FIGURE 19.** Path composition.

**FIGURE 20.** Incoming and outgoing edges.

c links this node directly, and trees a, b, and d approach it in two steps, as illustrated in Figure 21.

**Lemma 7:** The possibilities are limited to only five patterns: P1, P2, P3, P4, and P5 in the leaf portions of all ISTs in \( S_{n,k} \), as illustrated in Figure 22.

**Proof:**

**Pattern P1:**
Suppose the pattern of node \( v \) in \( T_x \) is P1. \( v \) is a leaf node traversed by \( T_x \) in the first execution of buildLeaf function, and \( v \)'s first symbol is \( x \).

**Pattern P2:**
Suppose the pattern of node \( v \) in \( T_x \) is P2. \( v \) is a leaf node traversed by \( T_x \) in the second execution of buildLeaf function. \( v \) can approach \( T_x \) by its children or friends whose first symbol is \( x \).

**Pattern P3:**
Suppose that node \( v \)'s last symbol is \( A \), and \( v \) wants a middle node \( m \) to approach the frame of Tree \( x \) through \( m \). However, the edge from \( m \) to \( v \) is used in the frame of Tree \( A \). We know that Tree \( n-k+2 \) does not conduct any buildFrame function and the last symbol of its frame nodes is the same as the last symbol \( n \) of the root. Thus, node \( v \) takes advantage of the symbol \( n-k+2 \) to find a node \( w \) according to Lemma 5. \( w \)'s first symbol is \( n-k+2 \). Finally, node \( w \) can use its symbol \( x \) to approach the frame of Tree \( x \).

P1, P2, and P3(type 1 and type 2) patterns are illustrated in the following examples.

In \( S_{n,4} \), the ISTs have five paths from the root to node 5431 as follows:

- \( 3456 \rightarrow 1456 \rightarrow 6451 \rightarrow 3451 \rightarrow 5431 \) (frame)
- \( 3456 \rightarrow 2456 \rightarrow 6452 \rightarrow 3452 \rightarrow 5432 \rightarrow 1432 \rightarrow 2431 \rightarrow 5431 \) (P2)
- \( 3456 \rightarrow 6453 \rightarrow 1453 \rightarrow 4153 \rightarrow 5143 \rightarrow 1543 \rightarrow 3541 \rightarrow 4531 \rightarrow 5431 \) (P3 type 1)
- \( 3456 \rightarrow 4356 \rightarrow 1356 \rightarrow 3156 \rightarrow 5136 \rightarrow 4136 \rightarrow 1436 \rightarrow 6431 \rightarrow 5431 \) (P2)
- \( 3456 \rightarrow 5436 \rightarrow 6435 \rightarrow 1435 \rightarrow 5431 \) (P1)

Node 5431’s last symbol is 1; hence, node 5431 is \( T_{1}^{6,4} \)'s frame node. In \( T_{1}^{6,4} \), node 3451 is node 5431’s parent, and node 3451 use 3 to visit node 5431. Therefore, \( T_{1}^{6,4} \) is not able to use 3 to visit node 5431. Finally, node 5431 uses 4 to reach the frame of \( T_{3}^{6,4} \).

In \( S_{n,4} \), the ISTs have five paths from the root to node 4351 as follows:

- \( 3456 \rightarrow 1456 \rightarrow 6451 \rightarrow 3451 \rightarrow 4351 \) (frame)
- \( 3456 \rightarrow 2456 \rightarrow 6452 \rightarrow 3452 \rightarrow 4352 \rightarrow 1352 \rightarrow 2351 \rightarrow 4351 \) (P2)
- \( 3456 \rightarrow 6453 \rightarrow 1453 \rightarrow 4153 \rightarrow 3154 \rightarrow 1354 \rightarrow 4351 \) (P3 type 2)
- \( 3456 \rightarrow 1456 \rightarrow 1356 \rightarrow 6351 \rightarrow 4351 \) (P2)
- \( 3456 \rightarrow 5436 \rightarrow 6435 \rightarrow 4135 \rightarrow 3145 \rightarrow 1345 \rightarrow 5341 \rightarrow 4351 \) (P2)

Note 4351’s last symbol is 1; hence, node 4351 is \( T_{1}^{6,4} \)'s frame node. In \( T_{1}^{6,4} \), node 4351 is node 4351’s parent, and node 4351 must use 3 to visit node 4351; Therefore, \( T_{3}^{6,4} \) is not able to use 3 to visit node 4351. Finally, node 4351 uses 4 to reach the frame of \( T_{3}^{6,4} \).

**Pattern P4:**

The concept of P4 is similar to that of P3. Suppose that node \( v \)'s last symbol is \( A \), and \( v \) wants a middle node \( m_1 \) to approach the frame of Tree \( x \) through \( m_1 \). However, the edge from \( m_1 \) to \( v \) is used in the frame of Tree \( A \). Thus, node \( v \) takes...
advantage of the symbol $n - k + 2$ to find a node $w$ according to Lemma 5. $w$'s first symbol is $n - k + 2$. Then, node $w$ wants to make node $m_2$ to approach the frame of Tree $x$ through $m_2$. However, the edge from $m_2$ to $w$ is still used in the frame of Tree A. Thus, node $w$ must make node $z$ according to Lemma 5. Notably, $w$'s first symbol is $n - k + 2$; therefore $z$'s last symbol is $n - k + 2$. Node $z$ must use its symbol $x$ to approach the frame of Tree $x$. Node $z$ only passes one node in cluster $n - k + 2$ to reach the frame of Tree $x$ according to Lemma 6.

The following example demonstrates P4 pattern. There are five paths from the root to node 6421 in $S_{6,4}$.

$3456 \rightarrow 1456 \rightarrow 6451 \rightarrow 2451 \rightarrow 5421 \rightarrow 6421$ (frame)

$3456 \rightarrow 2456 \rightarrow 6452 \rightarrow 5462 \rightarrow 1462 \rightarrow 2461 \rightarrow 6421$ (P2)

$3456 \rightarrow 6453 \rightarrow 2453 \rightarrow 5423 \rightarrow 1423 \rightarrow 3421 \rightarrow 6421$ (P2)

$3456 \rightarrow 4356 \rightarrow 1356 \rightarrow 3156 \rightarrow 2156 \rightarrow 5126 \rightarrow 4126 \rightarrow 1426 \rightarrow 6421$ (P1)

$3456 \rightarrow 5436 \rightarrow 6435 \rightarrow 4635 \rightarrow 2635 \rightarrow 3625 \rightarrow 4625 \rightarrow 5624 \rightarrow 1624 \rightarrow 4621 \rightarrow 6421$ (P4)

Node 4621’s parent in $T_{6,4}^1$ is node 5621, and node 5621’s first symbol is still 5. Hence, node 4621 in $T_{6,4}^1$ should use 4 to approach the frame of $T_{6,4}^1$ as follows.

$3456 \rightarrow 1456 \rightarrow 6451 \rightarrow 6451 \rightarrow 2451 \rightarrow 5621 \rightarrow 6421$ (frame)

$3456 \rightarrow 2456 \rightarrow 6452 \rightarrow 5462 \rightarrow 1624 \rightarrow 2461 \rightarrow 6421$ (P2)

$3456 \rightarrow 6453 \rightarrow 5436 \rightarrow 6453 \rightarrow 4635 \rightarrow 2635 \rightarrow 3625 \rightarrow 4625 \rightarrow 5624 \rightarrow 1624 \rightarrow 4621$ (P3 type 2)

**Pattern P5:**

P5 only occurs in Tree $n - k + 1$ for some nodes of the root cluster. The parents of those nodes belong to P4. For instance, in $S_{6,4}$, the paths from the root to node 5426 are as follows.

$3456 \rightarrow 1456 \rightarrow 5416 \rightarrow 2416 \rightarrow 1426 \rightarrow 5426$

$3456 \rightarrow 2456 \rightarrow 5426$

$3456 \rightarrow 6453 \rightarrow 5463 \rightarrow 2653 \rightarrow 5623 \rightarrow 4623 \rightarrow 3624 \rightarrow 5624 \rightarrow 1624 \rightarrow 4625 \rightarrow 5426$ (P5)

$3456 \rightarrow 4356 \rightarrow 3156 \rightarrow 3256 \rightarrow 4256 \rightarrow 5246 \rightarrow 2546 \rightarrow 4526 \rightarrow 5426$

$3456 \rightarrow 5436 \rightarrow 2436 \rightarrow 3426 \rightarrow 5426$

**Lemma 8:** No common node exists in both the frame portion of Tree A and the leaf portions of other trees in the paths from the root to node $v$ in all trees.

**Proof:** The leaf portion has five patterns: P1, P2, P3, P4, and P5 according to Lemma 7. For P1 and P2, suppose that Tree $x$ is going to traverse node $v$.

**P1:**

$v$’s first symbol is equal to $x$ and Tree $x$ can traverse $v$ by a friend operation in a step. Thus, no common node exists in both Tree A’s frame portion and Tree $x$’s leaf portion.

**P2:**

Tree $x$ traverses node $v$ through a middle node $w$ in cluster A. Because edge $w \rightarrow v$ is unused in Tree A’s frame and Tree $x$ can connect node $w$ by a friend operation in a step, there is no common node exists between Tree A’s frame portion and Tree $x$’s leaf portion.

**P3:**

Suppose that nodes are composed of $\{a, b, c, d, e, \ldots, A\}$, and we omit the rear portion. We exhaust the following three cases. Triangles represent complete graphs.

**Case 1: a complete graph**

Suppose that node ade requires a node of which the first symbol $c$ approaches Tree c’s frame, as illustrated in Figure 23(a). Because we have applied the MBFS traversal, no node such as node cde does not possess all children according to Lemma 4. Thus, it cannot occur that a common node appears in node cde.

**Case 2: child frame**

Node bde in cluster A requires the symbol $c$ to approach Tree c’s frame, but $c$ is used by node cde, as illustrated in Figure 23(b). Hence, node bde uses the symbol $b$ to walk to node dbe. Then, node dbe walks to node cbe, and node bde walks back to Tree c’s frame at the end. If a common node appeared in node cbe in Tree A’s frame, the path would exist: cbe $\rightarrow$ cbe $\rightarrow$ dce $\rightarrow$ dce (marked red lines). We apply the MBFS traversal in buildFrame functions. If the function has traversed node cbe, it would use the purple lines in lieu of the red lines and blue line to traverse node bde in the MBFS traversal. Thus, a common node does not exist due to the shortest path rule.

**Case 3: friend frame**

Node dbe in cluster A requires the symbol $b$ to approach Tree b’s frame, but $b$ is used by node bde, as illustrated in Figure 23(c). Hence, node dbe uses $d$ to walk to node ebd. Then, node ebd walks to node bed, and node dbe walks back to Tree b’s frame finally. If a common node

![Figure 23. P5 does not occur in frames.](image-url)
appeared in node bed in Tree A’s frame, the path would exist: bed→deb→ed→bde (marked red lines). We apply the MBFS traversal in buildFrame functions. If the function has traversed node bed, it would use the purple lines in lieu of the red lines and blue line to traverse node bde in the MBFS traversal. Thus, a common node does not exist due to the shortest path rule.

**P4:**

As depicted in Figure 22, P2 pattern does not appear in the frames, and \( T_{n-k+2} \) does not execute any buildFrame function. Hence, no common node would appear in the frame.

**PS:**

It belongs to some nodes of the root cluster in Tree \( n-k+1 \). Thus, no common node exists in the frame.

**Lemma 9:** The nodes of the root cluster must be traversed ultimately in \( T_{n-k+1} \).

**Proof:** The root cluster in all ISTs except for \( T_{n-k+1} \) originate from the previous trees. Because the root cluster’s incoming edges are not used and \( T_{n-k+1} \) has no previous tree, the root cluster must be traversed ultimately in \( T_{n-k+1} \). For instance, the five paths of \( T_{1}^{6,4} \) to \( T_{5}^{6,4} \) from the root to node 5426 as follows:

\[
\begin{align*}
3456 & \rightarrow 1456 \rightarrow 5416 \rightarrow 2416 \rightarrow 1426 \rightarrow 5426 \\
3456 & \rightarrow 2456 \rightarrow 5426 \\
3456 & \rightarrow 6453 \rightarrow 4653 \rightarrow 2653 \rightarrow 5623 \rightarrow 4623 \rightarrow 3624 \rightarrow 5624 \rightarrow 4625 \rightarrow 6425 \rightarrow 5426 \\
3456 & \rightarrow 4356 \rightarrow 2356 \rightarrow 3256 \rightarrow 4256 \rightarrow 2546 \rightarrow 4526 \rightarrow 5426 \\
3456 & \rightarrow 5436 \rightarrow 2436 \rightarrow 3426 \rightarrow 5426
\end{align*}
\]

Node 5426 in \( T_{1}^{6,4}, T_{2}^{6,4}, T_{3}^{6,4}, \text{and } T_{5}^{6,4} \) originate from the previous trees in \( S_{5,3} \). In \( T_{5}^{6,4} \), note 5426 must be traversed finally.

**Theorem 1:** The recursive algorithm can construct \( n-1 \) ISTs in \( S_{n,k} \).

**Proof:** We prove the theorem by mathematical induction. If we know \( S_{x,y} \), then we know \( S_{x+1,y+1} \). When \( y = 1 \) and \( x = n-k+1 \), a complete graph of \( n-k+1 \) nodes holds. There are \( n-1 \) ISTs as illustrated in Figure 12.

Suppose that \( y = k \) and \( x = n \) holds. When \( y = k+1 \) and \( x = n+1 \), \( S_{n+1,k+1} \) is made up of \( n+1 \) instances of \( S_{n,k} \). We partition nodes into clusters by the last symbol of each node. First, we copy \( T_{n}^{k} \) to \( T_{n+1}^{k+1} \), \( T_{n}^{k} \) to \( T_{n+k+1}^{k+1} \), \( T_{n}^{k} \) to \( T_{n+k+2}^{k+1} \), \( T_{n}^{k} \) to \( T_{n+k+3}^{k+1} \), \( T_{n}^{k} \) to \( T_{n+k+4}^{k+1} \), \( T_{n}^{k} \) to \( T_{n+k+5}^{k+1} \). Second, we find germs as explained in Table 3.

**TABLE 3. Germs in \( S_{n+1,k+1} \).**

| Tree id | Germ |
|---------|------|
| 1       | \((n+1)(n-k+1)(n-k+3)\ldots n\) |
| 2       | \((n+1)(n-k+2)(n-k+3)\ldots n-k\) |
| \ldots  | \ldots |
| n-k     | \((n+1)(n-k+2)(n-k+3)\ldots n(n-k)\) |
| n-k+1   | \((n+1)(n-k+3)\ldots n(n-k+1)\) |
| n-k+3   | \((n+1)(n-k+4)\ldots n(n-k+1)\) |
| n-k+4   | \((n+1)(n-k+5)\ldots n(n-k+1)\) |

Third, we use the buildFrame function to create frames. Fourth, we use buildLeaf functions to create leaf portions. For cluster \( n+1 \), all trees in \( S_{n,k} \) are independent and \( T_{n-k+1}^{n+1,k+1} \) traverses cluster \( n+1 \) through other clusters; hence, all paths of all trees from the root to cluster \( n+1 \) should be internally node-disjoint. For other clusters, each path is made up of frame and leaf portions. The frame portions are distinct and the leaf portions of all paths can be classified into one leaf pattern of P1, P2, P3, P4, and P5, which have no common node with the frame portion in accordance with Lemma 8. Thus, the \( n \) ISTs in \( S_{n+1,k+1} \) are independent, and the recursive algorithm can construct \( n-1 \) ISTs in \( S_{n,k} \) correctly by the induction hypothesis.

**Theorem 2:** \( n-1 \) independent spanning trees in \( S_{n,k} \) constructed by the parallel algorithm are independent.

**Proof:** The ISTs of \( S_{n-k+1} \) are constructed by hand, as presented in Figure 12. The buildFrame function is an instance of the buildFrameP function with tailIN set to 1. We can use tailIN to fix the last tailIN symbol(s) to reduce the dimensions of \( S_{n,k} \) and create all frames of ISTs from \( S_{n-k+2} \) to \( S_{n,k} \) simultaneously, whereas in the recursive algorithm the dimensions increase one in each iteration, and ISTs constructed in this iteration must be copied to ISTs next iteration. According to Algorithm 3, we create a FrameGermSet, and the germs of \( S_{n,k} \) are identical to that of \( S_{n-k+2} \) to \( S_{n,k} \) in the recursive algorithm. Therefore, the frames created by both algorithms are the same. The findParent function produces five leaf patterns displayed in Figure 22 as the buildLeaf function does. Thus, the ISTs of \( S_{n,k} \) constructed by the parallel algorithm and the recursive algorithm are the same. According to Theorem 1, the parallel algorithm can construct \( n-1 \) ISTs in \( S_{n,k} \) accurately.

**V. ANALYSIS AND COMPARISON**

We programmed in PHP and created illustrations in Graphviz. The test cases are presented in Table 4. We tested whether all paths in all trees were internally vertex-disjoint. The results proved that all paths in all trees were internally vertex-disjoint. Both algorithms are correct.

**TABLE 4. The cases tested.**

| \( n-k \) | Cases |
|----------|------|
| 1        | \( S_{1,3}, S_{2,3}, S_{3,4}, S_{4,5}, S_{5,6}, S_{6,7}, S_{7,8}, S_{8,9} \) |
| 2        | \( S_{1,3}, S_{2,3}, S_{3,4}, S_{4,5}, S_{5,6}, S_{6,7}, S_{7,8}, S_{8,9}, S_{10,9} \) |
| 3        | \( S_{1,2,3}, S_{2,3}, S_{3,4}, S_{4,5}, S_{5,6}, S_{6,7}, S_{7,8}, S_{8,9}, S_{10,9}, S_{10,7} \) |
| 4        | \( S_{1,2,3}, S_{2,3}, S_{3,4}, S_{4,5}, S_{5,6}, S_{6,7}, S_{7,8}, S_{8,9}, S_{10,6} \) |
| 5        | \( S_{1,2,3}, S_{2,3}, S_{3,4}, S_{4,5}, S_{5,6}, S_{6,7}, S_{7,8}, S_{8,9}, S_{10,5} \) |
| 6        | \( S_{1,2,3}, S_{2,3}, S_{3,4}, S_{4,5}, S_{5,6}, S_{6,7}, S_{7,8}, S_{8,9}, S_{10,4} \) |
| 7        | \( S_{1,2,3}, S_{2,3}, S_{3,4}, S_{4,5}, S_{5,6}, S_{6,7}, S_{7,8}, S_{8,9}, S_{10,3} \) |
| 8        | \( S_{1,2,3}, S_{2,3}, S_{3,4}, S_{4,5}, S_{5,6}, S_{6,7}, S_{7,8}, S_{8,9}, S_{10,2} \) |
| 9        | \( S_{1,2,3}, S_{2,3}, S_{3,4}, S_{4,5}, S_{5,6}, S_{6,7}, S_{7,8}, S_{8,9}, S_{10,1} \) |
| 10       | \( S_{1,2,3}, S_{2,3}, S_{3,4}, S_{4,5}, S_{5,6}, S_{6,7}, S_{7,8}, S_{8,9}, S_{10,0} \) |

Because all nodes and all directed edges are traversed once, the time complexity of \( S_{n,k} \) is the summation of the numbers of nodes and directed edges from \( S_{n-k+1} \) to \( S_{n,k} \). The number of nodes in \( S_{n,k} \) is \( \frac{n!}{(n-k)!} \) and the number of
TABLE 5. The comparison of recursive and parallel algorithms.

| Algorithm | Main function | Time Complexity | The number of processors used | Concept |
|-----------|---------------|-----------------|-------------------------------|---------|
| Recursive | buildFrameP  | $O(n \times \frac{n^2}{n-k})$ | 1 | Simple |
| Parallel  | buildFrameP  | $O(n \times \frac{n^2}{n-k})$ | $n-1$ | Complex |

VI. CONCLUSION

In this paper, we proposed one recursive algorithm and one parallel algorithm for constructing ISTs in $(n,k)$-star graphs. The recursive algorithm is a top-down approach, and the parent of a node in an IST is not determined by any rule. The parallel algorithm combines top-down and bottom-up approaches. In this paper, correctness, proofs, and time complexity analyses were presented for both algorithms. We also described PHP implementations of both algorithms. The test results for different fifty-nine cases are in Table 4. The results prove that all of the cases in Table 4 are ISTs for both algorithms. We conclude that our algorithms are not only correct but also efficient, and furthermore, the parallel algorithm is more efficient than the recursive algorithm. We hope that our contribution can aid in novel IST research.

REFERENCES

[1] F. Bao, Y. Fenyu, Y. Hamada, and Y. Igarashi, “Reliable broadcasting and secure distributing in channel networks,” in Proc. Int. Symp. Parallel Architect., Algorithms Netw. (I-SPAN), Taipei, Taiwan, Dec. 1997, pp. 472–478.
[2] J.-H. Chang and J. Kim, “Ring embedding in faulty (n,k)-star graphs,” in Proc. 5th Int. Conf. Parallel Distrib. Syst. (ICPADS), 2001, pp. 99–106.
[3] 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,” Appl. Math. Comput., vol. 268, pp. 489–495, Oct. 2015.
[4] 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,” J. Combinat. Optim., vol. 33, no. 3, pp. 956–967, Apr. 2017.
[5] J.-M. Chang, T.-J. Yang, and J.-S. Yang, “A parallel algorithm for constructing independent spanning trees in twisted cubes,” Discrete Appl. Math., vol. 219, pp. 74–82, Mar. 2017.
[6] Y. S. Chen and K. S. Tai, “A near-optimal broadcasting in (n,k)-star graphs,” in Proc. Int. Conf. Softw. Eng. Appl. Netw. Parallel Distrib. Comput. (ACIS), 2000, pp. 217–224.
[7] B. Cheng, J. Fan, X. Jia, and S. Zhang, “Independent spanning trees in crossed cubes,” Inf. Sci., vol. 233, pp. 276–289, Jun. 2013.
[8] B. Cheng, J. Fan, X. Jia, and S. Zhang, “Constructive algorithm of independent spanning trees on mobius cubes,” Comput. J., vol. 56, no. 11, pp. 1347–1362, Nov. 2013.
[9] J. Cheriyan and S. N. Maheshwari, “Finding nonseparating induced cycles and independent spanning trees in 3-connected graphs,” J. Algorithms, vol. 9, no. 4, pp. 507–537, Dec. 1988.
[10] C. Wei-Kuo and C. Rong-Jay, “The (n,k)-star graph: A generalized star graph,” Inf. Process. Lett., vol. 56, no. 5, pp. 259–264, Dec. 1995.
[11] W. K. Chang and R. J. Chen, “Topological properties of the (n,k)-star graph,” Int. J. Found. Comput. Sci., vol. 9, no. 2, pp. 235–248, 1998.
[12] S. Curran, O. Lee, and X. Yu, “Finding four independent trees,” SIAM J. Comput., vol. 35, no. 5, pp. 1025–1058, Jan. 2006.
[13] S.-Y. Hsieh and C.-I. Tu, “Constructing edge-disjoint spanning trees in locally twisted cubes,” Theor. Comput. Sci., vol. 410, nos. 8–10, pp. 926–932, Mar. 2009.
[14] H.-C. Hsu, Y.-L. Hsieh, J. J. M. Tan, and L.-H. Hsu, “Fault Hamiltonicity and fault Hamiltonian connectivity of the (n,k)-star graphs,” Networks, vol. 42, no. 4, pp. 189–201, Dec. 2003.
[15] H. C. Hsu, C. K. Lin, H. M. Huang, and L. H. Hsu, “The spanning connectivity of the (n,k)-star graphs,” Int. J. Found. Comput. Sci., vol. 17, pp. 415–434, Apr. 2006.
[16] J.-F. Huang, S.-S. Kao, S.-Y. Hsieh, and R. Klassing, “Top-down construction of independent spanning trees in alternating group networks,” IEEE Access, vol. 8, pp. 112333–112347, 2020.
[17] Z. Hussain, B. AliDaiwi, and A. Cerny, “Node-independent spanning trees in Gaussian networks,” J. Parallel Distrib. Comput., vol. 109, pp. 324–332, Nov. 2017.
[18] A. Itai and M. Rodeh, “The multi-tree approach to reliability in distributed networks,” Inf. Comput., vol. 79, no. 1, pp. 43–59, Oct. 1988.
[19] S.-S. Kao, J.-M. Chang, K.-J. Pai, and R.-Y. Wu, “Constructing independent spanning trees on bubble-sort networks,” in Proc. Int. Comput. Combinatorics Conf. (COCOON), Cham, Switzerland: Springer, vol. 10976, 2018, pp. 1–13.
[20] 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,” J. Combinat. Optim., vol. 38, no. 3, pp. 972–986, Oct. 2019.
[21] Y.-J. Liu, J. K. Lan, W. Y. Chou, and C. Chen, “Constructing independent spanning trees for locally twisted cubes,” Theor. Comput. Sci., vol. 412, no. 22, pp. 2237–2252, May 2011.
[22] A. A. Rescigno, “Vertex-disjoint spanning trees of the star network with applications to fault-tolerance and security,” Inf. Sci., vol. 137, nos. 1–4, pp. 259–276, Sep. 2001.
[23] Y. Wang, J. Fan, X. Jia, and H. Huang, “An algorithm to construct independent spanning trees on parity cubes,” Theor. Comput. Sci., vol. 465, pp. 61–72, Dec. 2012.
[24] Y. Wang, J. X. Fan, G. D. Zhou, and X. H. Jia, “Independent spanning trees on twisted cubes,” J. Parallel Distrib. Comput., vol. 72, pp. 58–69, Jan. 2012.
[25] J. Werapun, S. Intakosum, and V. Boonjing, “An efficient parallel construction of optimal independent spanning trees on hypercubes,” J. Parallel Distrib. Comput., vol. 72, no. 12, pp. 1713–1724, Dec. 2012.
[26] J.-S. Yang, S.-M. Tang, J.-M. Chang, and Y.-L. Wang, “Parallel construction of optimal independent spanning trees on hypercubes,” Parallel Comput., vol. 33, no. 1, pp. 73–79, Feb. 2007.
[27] J.-S. Yang, H.-C. Chan, and J.-M. Chang, “Broadcasting secure messages via optimal independent spanning trees in folded hypercubes,” Discrete Appl. Math., vol. 159, no. 12, pp. 1254–1263, Jul. 2011.
[28] A. Zehavi and A. Itai, “Three tree-paths,” J. Graph Theory, vol. 13, no. 2, pp. 175–188, Jun. 1989.
EDDIE CHENG received the Ph.D. degree in combinatorics and optimization from the University of Waterloo, Canada, in 1995. He is currently a Distinguished Professor of Mathematics with the Department of Mathematics and Statistics, Oakland University. He has served as the Chair of the Department from 2010 to 2013 and as the Acting Chair from January 2016 to April 2016. He has directed over 30 high school students in research projects. Some of them have advanced to semifinals and beyond in national competitions such as Siemens Competitions and the Intel Science Talent Search. For his contributions in teaching and mentoring, he received the 2009 Presidents Council State Universities of Michigan Distinguished Professor of the Year Award and the 2018 University of Waterloo Faculty of Mathematics Alumni Achievement Medal. He is currently on the editorial team of a number of journals, including Networks (Wiley), International Journal of Machine Learning and Cybernetics (Springer), Discrete Applied Mathematics (Elsevier), Journal of Interconnection Networks (World Scientific), International Journal of Computer Mathematics: Computer Systems Theory (Taylor & Francis), and International Journal of Parallel, Emergent and Distributed Systems (Taylor & Francis).

SUN-YUAN HSIEH (Senior Member, IEEE) received the Ph.D. degree in computer science from National Taiwan University, Taipei, Taiwan, in June 1998. He then served the compulsory two-year military service. From August 2000 to January 2002, he was an Assistant Professor with the Department of Computer Science and Information Engineering, National Chi Nan University. In February 2002, he joined the Department of Computer Science and Information Engineering, National Cheng Kung University, where he is currently a distinguished Professor and the Dean of Research. Recently, he joined the Center for Innovative FinTech Business Models. His current research interests include design and analysis of algorithms, fault-tolerant computing, bioinformatics, parallel and distributed computing, and algorithmic graph theory. He is a Fellow of the British Computer Society (BCS). He received the 2007 K. T. Lee Research Award, the President’s Citation Award (American Biographical Institute) in 2007, the Engineering Professor Award of Chinese Institute of Engineers (Kaohsiung Branch) in 2008, the National Science Council’s Outstanding Research Award in 2009, and the IEEE Outstanding Technical Achievement Award (IEEE Tainan Section) in 2011.

* * *