Received November 28, 2019, accepted December 14, 2019, date of publication December 26, 2019,
date of current version January 7, 2020.

Digital Object Identifier 10.1109/ACCESS.2019.2962549

Constructing Independent Spanning Trees on Pancake Networks

DUN-WEI CHENG[1], CHIH-TE CHAN[1], AND SUN-YUAN HSIEH[2,3], (Senior Member, IEEE)

[1]Department of Computer Science and Information Engineering, National Cheng Kung University, Tainan 701, Taiwan
[2]Department of Computer Science and Information Engineering, Institute of Medical Informatics, National Cheng Kung University, Tainan 701, Taiwan
[3]Center for Innovative FinTech Business Models, National Cheng Kung University, Tainan 701, Taiwan

Corresponding author: Dun-Wei Cheng (dunwei.ncku@gmail.com)

ABSTRACT For any graph G, the set of independent spanning trees (ISTs) is defined as the set of spanning trees in G. All ISTs have the same root, paths from the root to another vertex between distinct trees are vertex-disjoint and edge-disjoint. The construction of multiple independent trees on a graph has numerous applications, such as fault-tolerant broadcasting and secure message distribution. The pancake graph is a subclass of Cayley graphs and since Cayley graphs are crucial for designing interconnection networks, constructing ISTs on these graphs is necessary for many practical applications. In this paper, we propose algorithms for constructing ISTs on pancake graph. Examine the use of our algorithm for constructing ISTs on pancake graph in different dimensions. We also present proofs about the construction of ISTs on pancake graph to verify that the correctness of these algorithms.

INDEX TERMS Interconnection networks, independent spanning trees, pancake networks.

I. INTRODUCTION

In graph theory, the pancake graph is a type of Cayley graphs that encodes a certain abstract structure of an element group into a graph. For example, each vertex in the pancake graph can be used to represent a permutation of the element set. Let $P(n)$ (alternatively referred to as an n-pancake graph) denotes the structure of a graph in which each vertex represents a permutation of n elements from 1 to n; also know as an n-dimensional pancake graph. Moreover, the graph's edges are given between permutations transitive by prefix reversals; this means that when two permutations of n elements are generated through prefix reversals, then there exists an edge that is connected between these two vertices (e.g., an edge exists between (1, 2, 3, 4) and (3, 2, 1, 4) because the prefix of three elements is a reversal). The cardinality of the vertex set in $P(n)$ is $n!$, and the n-pancake graph is $n-1$ regular. Figure 1 presents the structure of $P(3)$ and $P(4)$. The pancake graph can be regarded as a model of interconnecting networks for a parallel computing structure [1], [3], [4], [14]. Considerable attention has been paid to the inherent symmetric and reversive properties of graphs such as their relative sparsity compared with hypercubes and the sub-logarithmic diameter that can be used to represent a delay in communication [10], [15].

The associate editor coordinating the review of this manuscript and approving it for publication was Kashif Saleem.

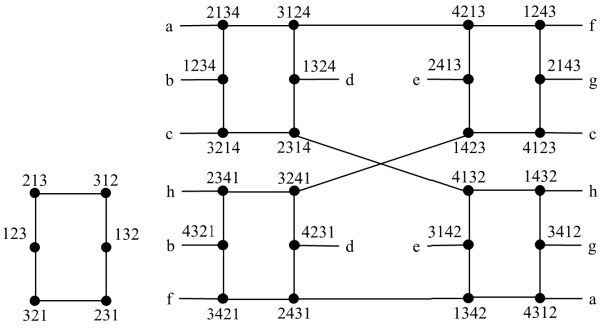


FIGURE 1. Example on $P(3)$ and $P(4)$.

In the design of interconnecting networks for parallel computing structures, fault-tolerance can be a crucial criterion that enables a system to continue operating properly in the event of failures either in hardware or software that relate to some components. Fault-tolerant broadcasting and secure message distribution have numerous applications for constructing multiple independent spanning trees (ISTs) in the relevant interconnecting networks. Let us consider the nature of ISTs. The ISTs of a given graph constitute a set of spanning trees that share the same root vertex, and each path from the root to the other vertices in the graph between each distinct spanning tree is vertex-disjoint and edge-disjoint. Because, in a given interconnecting network, the ISTs are vertex-disjoint and edge-disjoint. When some components

are no longer operating properly, the system can use a different spanning tree structure to broadcast and continue to complete jobs. Figure 2 illustrates the different spanning trees on $P(3)$. Considering spanning tree 1 and spanning tree 2 in Figure 2, we observe that the paths are vertex-disjoint and edge-disjoint. Considering spanning tree 3 and spanning tree 4 in Figure 2, we observe that although they are vertex-disjoint, the edge between root vertex $(1, 2, 3)$ and vertex $(2, 1, 3)$ is used in both trees; therefore, they are not independent [2], [9], [16].

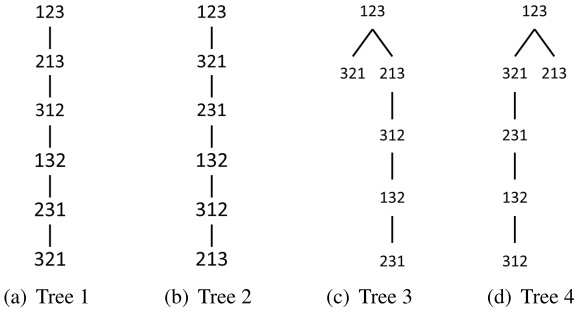


FIGURE 2. Example of spanning trees on $P(3)$.

Zehavi and Itai proposed that k ITSs can be constructed from a k-connected graph [19]. This conjecture has been confirmed on a k-connected graph when k is less than 5 (as demonstrated in [9], [7], [19], and [8] for $k = 2, 3$, and 4, respectively). However, the conjecture has yet to be confirmed for k greater than 5. Researchers have shifted their attention to methods for constructing ISTs on restricted interconnected networks [5], [6]. Numerous studies have been published regarding different types of hypercubes; in particular, star graphs can be used for constructing ISTs on different types of Cayley graphs [16]. We focus on constructing ISTs on different graphs [5], [6], [11]–[13], [18]; the emphasis is how to determine different independent paths from vertex v to the root vertex [10], [17]. We consider that an algorithm is fully parallelized if uses all network vertices for computation [5], [18]. In this paper, we propose parallel algorithms for determining the parent vertex for each vertex to construct ISTs.

Property 1: The greatest number of ISTs in $P(n)$ is $n - 1$.

Proof: $P(n)$ is $n - 1$ regular. Each vertex in a spanning tree has only one parent. Therefore, we can assume that the vertices in ISTs must connect to a different parent, and $n - 1$ vertices can be selected as the parent for a vertex. Hence, the greatest number of ISTs in $P(n)$ is $n - 1$.

II. PRELIMINARIES

Let $\pi_n(s)$ denotes the set of all permutations that conform to the following rules regarding variables n and s. The variable n is an integer, which represents the cardinality of elements. The permutations of n elements from 1 to n, and the suffix of these permutations are the same specific sequence s. When s is a null sequence, the permutation set is all permutations from 1 to n ($\pi_3 = (1, 2, 3)$, $(2, 1, 3)$, $(1, 3, 2)$, $(3, 1, 2)$, $(2, 3, 1)$, $(3, 2, 1)$). And the permutation set $\pi_3(3, 1)$ means the specific permutation set $(2, 3, 1)$ in which the suffix of every permutation is obeyed the s suffix sequence.

In the pancake graph, each vertex $u \in \pi_n$ represents a certain permutation of n elements. For permutation $u = (u_1, u_2, \ldots, u_n)$, let $u(i) = u_i$ denotes the value of the ith element in this permutation; and let $u^{-1}(u_i) = i$ denotes the position index of the element u_i in u. $u\langle i\rangle$ describes how to flip the ordering from the permutation u. Start from the element u_i and go back through u_1, after this, we concatenate the rest of permutation from u_{i+1} to u_n. That means the new permutation $u\langle i\rangle = (u_i, u_{i-1}, \ldots, u_2, u_1, u_{i+1}, \ldots, u_n)$.

We construct an n-dimensional pancake graph $P(n)$ using the vertex set $V(P(n)) \in \pi_n$ and connecting these vertices by the edge set $E(P(n)) = \{u, u\langle i\rangle | 2 \leq i \leq n\}$. The pancake graph is an undirected graph. Let P_i^n denotes a subset of vertices that the value of the last element equals to i. In this paper, we choose a particular subset of vertices for constructing distinct ISTs. And according to different subset we used, we denote IST as T_t^n which is constructed from the subset of vertices P_{t+1}^n. In the context of Property 1, we can note that the greatest number of ISTs in $P(n)$ is $n - 1$. We named this subset that we used for constructing IST as "Main Subset" and the rest of subsets as "Other Subsets". These two terms are crucial for the following descriptions.

III. ALGORITHM OF ISTs

This section presents the proposed algorithms, designing two strategies for constructing $n - 1$ ISTs on n-dimensional pancake graph $P(n)$. The first strategy constructs $n - 2$ ISTs according to "Main Subset" that introduce previously. The second strategy deals with the exceptional case and find "the last spanning tree" on $P(n)$. Figure 3 sketches the structure of our ISTs and uses the circle to represent the vertex subset. Since the pancake graph is symmetric, w.l.o.g., this paper using the vertex with permutation $(1, 2, \ldots, n)$ as the root for every ISTs.

- **The First Strategy:** Constructing the IST T_t^n ($1 \leq t \leq n - 2$) according to "Main Subset" P_{t+1}^n. Start from the root and construct a spanning tree in P_n^n. Extending this spanning tree to "Main Subset" P_{t+1}^n. After that, connecting the remaining vertices in "Other Subsets".
- **The Second Strategy:** Constructing the "last spanning tree" T_{n-1}^n. Start from the root and construct a spanning tree in P_1^n. Extend the last spanning tree to subsets P_i^n ($2 \leq i < n$). After that, connecting the remaining vertices in subset P_n^n.

During the construction, since each vertex in the spanning tree has only one parent vertex but might have multiple children. Instead of constructing through the child vertices, we focus on how to choose the one parent for each vertex in this paper.

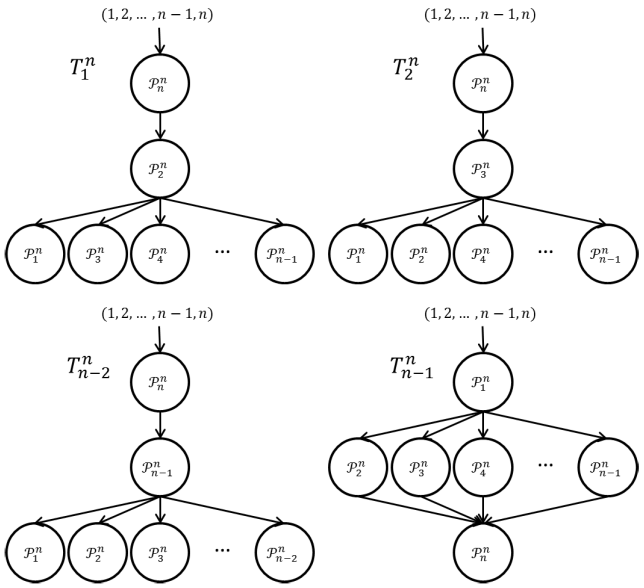


FIGURE 3. Structure of ISTs.

A. MAIN ALGORITHM

The main algorithm constructs ISTs which connecting vertices in P_n^n (the specific vertex subset that the last element of each vertex equals to n), and then using two subroutines to deal with vertices in "Main Subset" and "Other Subsets". According to our first strategy, we construct $n-2$ ISTs on an n-pancake graph and the last spanning tree will be constructed by those unused connections. We require four parameters to present the construction of ISTs: the input vertex u, the position index *pos*, the ordinal number of ISTs t, and a result table *table* to record the constructions.

- u: an input vertex in $P(n)$ which represents a permutation, $u = (u_1, u_2, \ldots, u_n)$.
- *pos*: denote a position index which indicates the inspecting element.
- t: the ordinal number of ISTs that we construct here, $t = 1, .., n-1$.
- *table*: record the construction of all ISTs. E.g. $table[u, p] = t$ means that the connection between vertex u and vertex $u\langle p\rangle$ is used by IST t.

Since the pancake graph is symmetric and hierarchy, the structure of vertex in P_n^n is homogeneous to the $(n-1)$-dimensional pancake subgraph $P(n-1)$. The only difference is that the permutation of a vertex in P_n^n gets one more element at the end and its value equals to n, as we can see the structures of P_4^4 and $P(3)$ in Figure 1. Therefore, we can consider P_n^n to be the same as $P(n-1)$, and follow the ISTs in $P(n-1)$ to construct the spanning trees in P_n^n.

According to the hierarchical structure, we inspect the input vertex u recursively in the n-pancake subgraph until it does not belong to $P_i^i (1 < i \leq n)$. This means at position index *pos*, this input vertex is no longer following the decreasing order (e.g. (2, 1, 3, 4) will leave the recursion when $pos = 2$). Start from how to connect the root vertex, w.l.o.g., let $(1, 2, 3, \ldots, n)$ be the root for each ISTs. When the input vertex satisfies that $u\langle t+1\rangle$ equals to the permutation of root, we choose the root vertex as a parent for this input vertex in T_t^n. And then, the main algorithm inspects whether the input vertex belongs to vertex subset P_n^n. When position index *pos* is less than $t+1$, the main algorithm assigns this input vertex to its parent vertex $u\langle t+1\rangle$.

Algorithm 1 Main Algorithm

Input : u: the input vertex;
pos: the position index we inspecting;
t: the ordinal number of a spanning tree;
table: a two dimensional array records the constructions of $n-1$ ISTs

Output: the parent of this vertex in this spanning tree and the two dimensional array marking which parent of the vertices is used

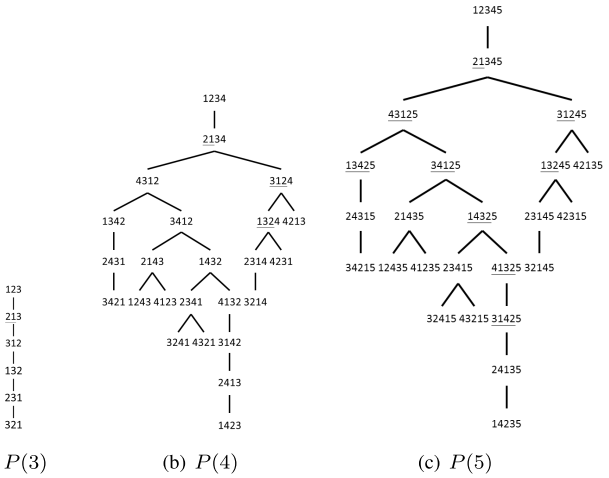
```
table is initailize to 0
if u(pos) = pos then
    Main Algorithm(u, pos − 1, t, table)
else if u⟨t + 1⟩ = root then
    table[u, t + 1] = t        ◁ (1)
else if pos < t + 1 then
    table[u, t + 1] = t        ◁ (2)
else if pos = t + 1 then
    for i = 2; i ≤ pos; i = i + 1 do
        if table[u, i] = 0 then
            table[u, i] = t    ◁ (3)
else if u⟨pos⟩ ∈ π_n(1, t + 2, t + 3, ..., n) then
    table[u, pos] = t          ◁ (4)
else if
u⟨pos⟩ ∈ {π_n(t + 1, t + 3, t + 4, ..., n) ∪ π_n(t + 1, n)}
then
    table[u, pos] = t          ◁ (5)
else if u(1) = t + 1 then
    table[u, pos] = t          ◁ (6)
else if u(pos) = t + 1 then
    Goto Algorithm 2
else
    Goto Algorithm 3
```

In vertex subset P_n^n, we also indicate two good vertex sets which are used to directly connect to "Main Subset" and "Other Subsets":

- The permutation belongs to $\pi_n(1, t+2, t+3, \ldots, n)$: There is a suffix of this permutation that starts from 1, and attaches the increasing sequence from $t+2$ to n (e.g. for IST $t = 2$, (2, 3, 1, 4, 5, 6) and (3, 2, 1, 4, 5, 6) are first kind of good vertices in P_6^6).
- And the second kind of good vertex: there is a suffix that starts from $t+1$, and attaches the increasing sequence

from j to n for $j \in [t+3, n]$ (e.g. $\pi_n(t+1, t+3, t+4, \ldots, n), \pi_n(t+1, t+4, \ldots, n), \ldots, \pi_n(t+1, n)$).

When the input vertex in "Main Subset" or "Other Subsets" that flipping at the *pos*-th element will become a permutation belongs to good vertex sets, we choose this good vertex as the parent for this input vertex. And by this connection, we extend the construction of IST T_t^n to "Main Subset" and "Other Subsets". Figure 4 presents an example of different pancake graphs in T_1^n; vertices with a bottom line can be considered good vertices in this spanning tree.



(a) $P(3)$ (b) $P(4)$ (c) $P(5)$

FIGURE 4. Find good vertex set on different pancake graph in T_1^n.

The next step, we choose a parent vertex in "Main Subset" for the input vertex u whose first element equals to $t+1$. The parent vertex we chose represents the permutation of u that flips at the *pos*-th element and connect a vertex in "Main Subset" to a vertex in "Other Subsets" for T_t^n. The last part of the main algorithm, we deal with the rest of vertices. When the input vertex is determined to end on "Main Subset", we choose a parent for this vertex through subroutine "Algorithm 2". And other vertices must be sent to "Algorithm 3".

B. ALGORITHM 2

Algorithm 2 serves to connect vertices in "Main Subset" P_{t+1}^n for constructing the specific IST T_t^n. It requires: the input vertex u, the position index *pos*, the ordinal number of ISTs t, and result table *table* to record the constructions. Here, we use one more parameter *Hubs* to construct IST in "Main Subset". *Hubs* is a set of vertices in "Main Subset" which flips at the same position index *pos* from the first kind of good vertices in P_n^n. We denote these vertices as hubs in "Main Subset".

Similar to the main algorithm, we keep shrinking pancake graph in "Main Subset" down to the smaller pancake subgraph that contains at least one hub. We introduce a new notation $Hubs(d)$ to denote an element set which collects the dth element for each hub vertex in the *Hubs* (e.g. the *Hubs* is (1, 2, 3, 4), (2, 4, 3, 1), then $Hubs(4) = \{4, 1\}$). Algorithm 2 recursively trace those hub vertices by finding the longest suffix between the input vertex and any hub. And let position index d to indicate the prefix sequence besides its longest common suffix.

Algorithm 2 Find ISTs Algorithm on Main Subset

Input : u: the input vertex;
pos: the position index we inspecting;
t: the ordinal number of a spanning tree;
table: a two dimensional array records the constructions of $n-1$ ISTs;
Hubs: the vertex set connects to good vertices
Output: the parent of this vertex in this spanning tree and the two dimensional array marking which parent of the vertices is used

```
d = pos
if u(d) ∈ Hubs(d) then
    l ← ∅
    for i = 1; i ≤ |Hubs|; i = i + 1 do
        if u(d) = Hubs[i](d) then
            l ← Hubs[i]
    Find ISTs Algorithm on Main Subset(u, d − 1, t, table, l)
else if u(d) = max_{1≤s≤d} u(s) then
    table[u, d] = t                                   ◁ (1)
else if u(1) ∈ Hubs {d} then
    table[u, d] = t                                   ◁ (2)
else if u(1) = min_{1≤s≤d} u(s) then
    table[u, d] = t                                   ◁ (3)
else if u(1) = max_{1≤s≤d−1} u(s) then
    table[u, u^{-1}(min_{1≤s≤d−1} u(s))] = t           ◁ (4)
else
    table[u, u^{-1}(max_{1≤s≤d−1} u(s))] = t           ◁ (5)
```

In each "Main Subset", there is at least one hub vertex which directly connects to a good vertex in P_n^n and we use this connection to construct the IST. Specifically, vertices in "Main Subset" must find a path to a hub according to its prefix sequence. To demonstrate the operation of Algorithm 2, Figure 5 and Figure 6 show an example of constructing IST T_2^5 in P_3^5. The cycle in these figures represents the hub vertex and the underlined sequence denotes the prefix sequence that we use to choose the parent for this vertex. Dotted arrows in these figures indicate which vertex is inspected and which vertex is the parent. Corresponding to the rules in Algorithm 2, the element in boldface constitutes the key element to choosing these rules. According to the rules, we choose the parent for each vertex in "Main Subset" and find the path to connect to the hub for constructing the IST.

C. ALGORITHM 3

Algorithm 3 pertains to constructing a spanning tree in "Other Subsets". When the first element of input vertex u

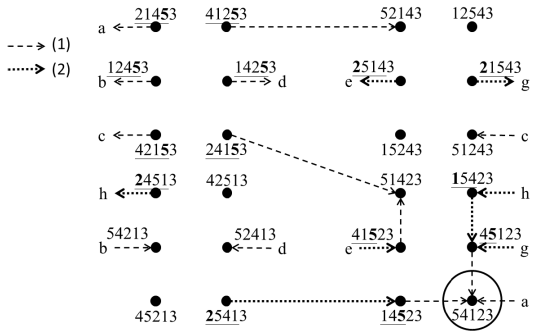


FIGURE 5. The example for the rule from (1) to (2) in Algorithm 2.

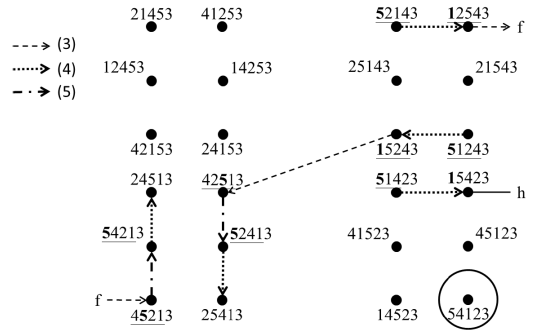


FIGURE 6. The example for the rule from (3) to (5) in Algorithm 2.

equals to the main subset number $m = t+1$, this vertex would connect to "Main Subset" by flipping the whole permutation. For the rest of vertices in "Other Subsets", we flip it at the element whose value equals to the main subset number and assigning that vertex as its parent. However, as discussed in the context of Algorithm 2, some rules would entail conflicts in "Other Subsets". To maintain Algorithm 2, special rules must be determined to find the new parents in these conflicts.

As mentioned, we must add some exceptions; therefore, we design a pre-treatment for inspecting whether we must use special rules to find the parent for this input vertex. We introduce two variables:

- *gvs*(**an abbreviation of "good vertex set"**)**:** *gvs* stores an integer to denote the main subset number for which IST that the second kind of good vertex would directly connect to this input vertex.
- *check*(**an abbreviation of "check position index"**)**:** *check* stores an integer to denote which position index that we should inspect when this input vertex be treated as a vertex in "Main Subset" for other ISTs.

Algorithm 3 pertains to inspecting and addressing exceptions. Because the outlined rules pertain to Algorithm 2, we can review them in Algorithm 2 to easily determine them.

For example, the rule (1) in Algorithm 3 is corresponding to the rule (3) in Algorithm 2. We inspect this input vertex from the first element to the check position. And when the first element is the minimum between this prefix, Algorithm 2 assigns this input vertex to the parent which flipped at the check position when we treat this vertex belongs to "Main Subset" in other IST. But when the value of the element at the check position is equal to $t + 1$ this connection conducts a conflict since we also intend to assign to the same parent in Algorithm 3. The difference is that Algorithm 2 treats this input vertex as a vertex in "Main Subset", but in Algorithm 3 it is a vertex belongs to "Other Subsets". To overcome this conflict, we choose a new parent for this input vertex when we treat it as a vertex belonging to "Other Subsets". Flipping at the position whose value is the maximum between the prefix from the first element to the *check* position. The rest of rules address the exceptions according to the rule (2) and rule (4) in Algorithm 2.

Algorithm 3 Find ISTs Algorithm on Other Subcases

Input : u: the input vertex;
pos: the position index we inspecting;
t: the ordinal number of a spanning tree;
table: a two dimensional array records the constructions of $n-1$ ISTs

Output: the parent of this vertex in this spanning tree and the two dimensional array marking which parent of the vertice is used

$gvs = 0$
$check = 0$
for $i = 1$; $i \leq pos \cap u(1) = pos$; $i = i + 1$ **do**
  if $t + 1\text{-}i \neq u(i)$ **then**
    $gvs = u(i)$ **break**
for $i = pos - 1$; $i > 2$; $i = i - 1$ **do**
  if $i > pos - u(pos) + 1$ **then**
    if $u(i) > u(pos) \cup u(i) = 1$ **then**
      $check = i$ **break**
  else if $i = pos - u(pos) + 1$ **then**
    if $u(i) \neq 1$ **then**
      $check = i$ **break**
  else
    if $u(i) > pos - i + 1$ **then**
      $check = i$ **break**
if $u(1) = \min_{1 \leq s \leq check} u(s) \cap u(check) \neq \max_{1 \leq s \leq check} u(s) \cap u(check) = t + 1$ **then**
  $table[u, u^{-1}(\max_{1 \leq s \leq check} u(s))] = t$ ◁ (1)
else if $u(check) \neq \max_{1 \leq s \leq check} u(s) \cap u(1) = \max_{1 \leq s \leq check-1} u(s) \cap \min_{1 \leq s \leq check-1} u(s) = t + 1$ **then**
  $table[u, u^{-1}(gvs)] = t$ ◁ (2)
else if $check > pos - u(pos) + 1 \cap u(1) < u(pos) \cap u(1) > 1 \cap u(check) = t + 1$ **then**
  $table[u, u^{-1}(\max_{1 \leq s \leq check-1} u(s))] = t$ ◁ (3)
else
  $table[u, u^{-1}(t + 1)] = t$ ◁ (4)

IV. PROOF REGARDING ALGORITHMS

We prove some of the rules of the aforementioned algorithms to ensure that we can construct ISTs on the n-pancake graph ($P(n)$).

Property 2: $P(n-1)$ is the graph that each vertex removes the last element n on P_n^n.

Proposition 1: According to the structure of ISTs. The root connects to the subset P_1^n at the vertex $(n, n-1, \ldots, 2, 1)$ in the last spanning tree T_{n-1}^n, and other vertices have paths to this subset. Therefore, the vertices whose first element is one connect to P_1^n.

Proposition 2: The spanning tree T_i^n on P_n^n is the same as the spanning tree T_i^{n-1} on $P(n-1)$ $(1 \leq i \leq n-2)$.

Proof: According to Property 2, $P(n-1)$ is the graph that each vertex removes the last element n on P_n^n. Therefore, the spanning trees that added an element n to all vertices on $P(n-1)$ are also the spanning trees on $P(n)$. Hence, the spanning tree T_i^n on P_n^n is the same as the spanning tree T_i^{n-1} on $P(n-1)$ $(1 \leq i \leq n-2)$.

Lemma 1: When there are $n-2$ ISTs, the graph g is induced by the rule of the last spanning tree on $P(n)$. P_1^n is connected.

Proof: In other spanning trees, the vertices in P_1^n could find the parent by Algorithm 3. According to Algorithm 3, when vertices find their parents, they reserve a neighbor for Algorithm 2 to ensure Algorithm 2 works. Then, according to Algorithm 2, each vertex must have a path to a hub. When Algorithm 3 works on P_1^n, the second *FOR* loop in Algorithm 3 finds only one vertex $(n, n-1, \ldots, 2, 1)$ as common ancestor for the last spanning tree. In other words, each vertex in P_1^n reserves a neighbor to ensure a path to vertex $(n, n-1, \ldots, 2, 1)$ on Algorithm 3. Therefore, any two vertices are connected by the vertex $(n, n-1, \ldots, 2, 1)$, and P_1^n is connected in g.

Lemma 2: When there are $n-2$ ISTs, the graph g is induced by the rule of the last spanning tree on $P(n)$.

Proof: According to Proposition 1, the vertices whose first element is one connect to P_1^n. Let v denotes the vertex whose first element is one. Then, if other vertices have paths to v, these vertices have paths to P_1^n. Since the rule 4 in Algorithm 2, there are two different cases to find paths to v. The first case focuses on vertices which the rule 4 in Algorithm 2 doesn't work in other spanning tree, and the second case focuses on remain vertices.

Case 1: According to the structure of ISTs, Algorithm 2 and Algorithm 3. P_i^n $(2 \leq i \leq n-1)$ is the Main Subset in different spanning trees. Then, most vertices don't connect to v in other spanning trees, therefore we assign v to be their parent in the last spanning tree.

Case 2: According to the rule 4 in Algorithm 2, some exceptional vertices connect to v in other spanning trees, and these vertices can not connect to v this time. However, since the rule of edge, if the rule 4 in Algorithm 2 works on this vertex, the rule must not be used on the neighbors of this vertex. These exceptional vertices can connect to a neighbor which is not used in other spanning trees. This neighbor has a path to v by **Case 1**, these exceptional vertices have paths to P_1^n.

Hence, other vertices have paths to P_1^n.

Lemma 3: When there are $n-2$ ISTs, the graph g is induced by the rule of the last spanning tree on $P(n)$. There are $n!$ vertices in g, and g is connected.

Proof: According to the structure of ISTs. The root connects to the subset P_1^n in the last spanning tree T_{n-1}^n at the vertex $(n, n-1, \ldots, 2, 1)$, and this subset connects to other subsets. According to Lemma 1, P_1^n is connected in g. And according to Lemma 2, other vertices have paths to P_1^n. Hence, there are $n!$ vertices in g, and g is connected.

Lemma 4: When there are $n-2$ ISTs, the graph g is induced by the rule of the last spanning tree on $P(n)$. There are $n! - 1$ edges in g.

Proof: Because there are $n-2$ ISTs, each vertex choose only one neighbor as parent in the last spanning tree except for the root. Then, there are $n! - 1$ edges in g.

Theorem 1: When there are $n-2$ ISTs, the graph g is induced by the rule of the last spanning tree on $P(n)$. g is a tree.

Proof: According to Lemma 3 and Lemma 4. There are $n!$ vertices in g, there are $n!-1$ edges in g, and g is connected. Hence, g is a tree.

Theorem 2: The algorithm in this paper constructs $n-1$ ISTs on $P(n)$ $(n \in \mathbb{N})$.

Proof:

Basic Step: The algorithm constructs 0 ISTs on $P(1)$.

Induction Step: According to Proposition 2, the algorithm constructs ISTs on P_n^n by following ISTs on $P(n-1)$.

By induction hypothesis, the algorithm in this paper can construct $n-1$ ISTs on $P(n)$ $(n \in \mathbb{N})$.

V. CONCLUSION

The pancake graph is notable and similar to a bubble sort graph. Both the pancake and bubble sort graphs can have $n!$ vertices if they are constructed from the same value. However, the rules for edges are different. In the pancake graph, two different vertices u, v in the pancake have an edge between them, if for an integer $i(2 \leq i \leq n)$, u and v are the same when u flips its sequence on index i. However, the two different vertices u, v in a bubble sort graph have an edge between them, if for two integers $i, j(1 \leq i < j \leq n)$, u and v are the same, when we switch symbols in index i, j. Because of this difference between the pancake graph and bubble sort graph, we cannot use the rules for constructing ISTs in bubble sort.

In this paper, we propose feasible algorithms for constructing ISTs on the pancake graph. According to the first kind of structures, we construct $n-2$ ISTs on $P(n)$. According to the second kind of structures and the rule for last spanning

tree, we can construct one IST. Then, we construct $n-1$ ISTs on $P(n)$.

According to algorithms, input a vertex to find its parent in ISTs by its sequence. Then we consider that is parallel computing and the time complexity is $O(n \times n!)$ on $P(n)$. $n!$ means the size of pancake graph. Then, we can consider that our algorithm is in polynomial time. We prove the effectiveness of the algorithms in constructing ISTs rapidly and easily. We hope these theoretical contributions can be useful for other problems.

REFERENCES

[1] S. G. Akl, K. Qiu, and I. Stojmenović, "Fundamental algorithms for the star and pancake interconnection networks with applications to computational geometry," *Networks*, vol. 23, no. 4, pp. 215–225, Jul. 1993.

[2] F. Bao, Y. Funyu, Y. Hamada, and Y. Igarashi, "Reliable broadcasting and secure distributing in channel networks," *IEICE Trans. Fundam. Electron., Commun. Comput. Sci.*, vol. 81, no. 5, pp. 796–806, 1998.

[3] D. W. Bass and I. Sudborough, "Pancake problems with restricted prefix reversals and some corresponding Cayley networks," *J. Parallel Distrib. Comput.*, vol. 63, no. 3, pp. 327–336, Mar. 2003.

[4] P. Berthome, A. Ferreira, and S. Perennes, "Optimal information dissemination in star and pancake networks," *IEEE Trans. Parallel Distrib. Syst.*, vol. 7, no. 12, pp. 1292–1300, 1996.

[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.-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.

[7] J. Cheriyan and S. Maheshwari, "Finding nonseparating induced cycles and independent spanning trees in 3-connected graphs," *J. Algorithms*, vol. 9, no. 4, pp. 507–537, Dec. 1988.

[8] S. Curran, O. Lee, and X. Yu, "Finding four independent trees," *SIAM J. Comput.*, vol. 35, no. 5, pp. 1023–1058, Jan. 2006.

[9] 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.

[10] K. Kaneko and S. Peng, "Disjoint paths routing in pancake graphs," in *Proc. IEEE 7th Int. Conf. Parallel Distrib. Comput., Appl. Technol. (PDCAT)*, Dec. 2006, pp. 254–259.

[11] 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.

[12] 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. Combinat. Conf.* Cham, Switzerland: Springer, 2018, pp. 1–13.

[13] S.-S. Kao, J.-M. Chang, K.-J. Pai, J.-S. Yang, S.-M. Tang, and R.-Y. Wu, "A parallel construction of vertex–disjoint spanning trees with optimal heights in star networks," in *Proc. Int. Conf. Combinat. Optim. Appl.* Cham, Switzerland: Springer, Dec. 2017, pp. 41–55.

[14] Q. T. Nguyen and S. Bettayeb, "On the genus of pancake network," *Int. Arab J. Inf. Technol.*, vol. 8, no. 3, pp. 289–292, 2011.

[15] M. J. Quinn, *Parallel Computing: Theory and Practice*. McGraw-Hill, 1994.

[16] 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.

[17] Y. Suzuki and K. Kaneko, "An algorithm for node-disjoint paths in pancake graphs," *IEICE Trans. Inf. Syst.*, vol. 86, no. 3, pp. 610–615, 2003.

[18] 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," *J. Supercomput.*, vol. 71, no. 3, pp. 952–965, Mar. 2015.

[19] A. Zehavi and A. Itai, "Three tree-paths," *J. Graph Theory*, vol. 13, no. 2, pp. 175–188, 1989.

DUN-WEI CHENG received the M.S. degree from the Department of Computer Science and Information Engineering, National University of Kaohsiung, in 2011. He is currently pursuing the Ph.D. degree with the Department of Computer Science and Information Engineering, National Cheng Kung University, Tainan, Taiwan. His current research interests include financial computing, artificial intelligence, system-level fault diagnosis, and hub location problem.

CHIH-TE CHAN received the M.S. degree from the Department of Computer Science and Information Engineering, National Cheng Kung University, Taiwan, in 2014. His current research interests include design and analysis of algorithms, and bioinformatics.

SUN-YUAN HSIEH 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. He is currently a Distinguished Professor and the Dean of Research. Recently, he also joints 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 from American Biographical Institute, in 2007, the Engineering Professor Award of Chinese Institute of Engineers from the Kaohsiung Branch, in 2008, the National Science Council's Outstanding Research Award, in 2009, and the IEEE Outstanding Technical Achievement Award from the IEEE Tainan Section, in 2011.