Constructing Independent Spanning Trees on Transposition Networks

CHIEN-FU LIN¹, JIE-FU HUANG², AND SUN-YUAN HSIEH³,⁴,⁵, (Senior Member, IEEE)

¹Department of Computer Science and Information Engineering, National Cheng Kung University, Tainan 701, Taiwan
²Institute of Medical Informatics, National Cheng Kung University, Tainan 701, Taiwan
³Institute of Manufacturing Information and Systems, National Cheng Kung University, Tainan 701, Taiwan
⁴International Center for the Scientific Development of Shrimp Aquaculture, National Cheng Kung University, Tainan 701, Taiwan
⁵Center for Innovative FinTech Business Models, National Cheng Kung University, Tainan 701, Taiwan

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

ABSTRACT In interconnection networks, data distribution and fault tolerance are crucial services. This study proposes an effective algorithm for improving connections between networks. Transposition networks are a type of Cayley graphs and have been widely used in current networks. Whenever any connection node fails, users want to reconnect as rapidly as possible, it is urgently in need to construct a new path. Thus, searching node-disjoint paths is crucial for finding a new path in networks. In this article, we expand the target to construct independent spanning trees to maximize the fault tolerance of transposition networks.

INDEX TERMS Independent spanning trees, transposition networks, interconnection networks, Cayley graphs.

I. INTRODUCTION

In modern, networks have extensively and progressively increased in speed. Enterprises are concerned with user experience. Moreover, stability is a growing concern. Once a connection node, such as a router, has failed, a network’s most crucial problem is to use some other path to restore connectivity. Node-disjoint paths can be found to solve such problems. Whenever a node fails on the path to a source node, systems can rapidly discover a new path to the source node. However, every node may connect to several node-disjoint paths, elegant and inelegant paths may overlap. As long as a node has failed, the records of other nodes may be invalid, and thus, some living nodes may be unsearchable. Independent spanning trees can be constructed to solve network connectivity problems.

A set of spanning trees in a graph \( G \) are vertex (resp., edge) independent if they are rooted at the same vertex \( r \), and for each vertex \( v \in V(G) \backslash \{r\} \), the paths from \( v \) to \( r \) are vertex (resp., edge) disjoint [20]. Independent spanning trees guarantee that for every node there exists a path to all other nodes in different node-disjoint spanning trees. The strategy of constructing multiple independent spanning trees in networks can be used for fault-tolerant broadcasting and secure distribution [2], [14]. Once we construct \( k \) independent spanning trees, we can obtain \( k \) node-disjoint paths for a node. These independent spanning trees can ensure fault tolerance, and this is because such a network can survive with \( k - 1 \) faulty components.

In the past twenty years, the IST problem has been solved on several interconnection networks, including chordal rings [15], twisted cubes [3], [27], cross cubes [7], Moebius cubes [8], locally twisted cubes [5], [12], [23], parity cubes [4], [26], hypercubes [28], [30], folded hypercubes [32], star networks [17], Gaussian networks [13], bubble-sort networks [18], [19], and recursive circulant graphs with \( G(cd^m) \) with \( d > 2 \) [31].

A Cayley graph \( \Gamma \) [1], is a graph that satisfies \( \Gamma = (G, S) \), for \( G \) is a finite group of \( V(\Gamma) \) and \( S \) is a subset of \( G \) with \( E(\Gamma) = \{(g, sg) | g \in G, s \in S\} \) [29]. The transposition network \( [6], [9]–[11], [16], [25] \) is a subclass of Cayley graphs. An \( n \)-transposition network refers to a network whose nodes are permutations of \( n \) symbols, and a node is adjacent to another node with an address that differs to arbitrary two-digit transposition. According to the properties of Cayley graphs, an \( n \)-transposition network is a symmetric graph [1]. Consider to symmetry properties of transposition networks, such a network has a part of structure similar to a bubble-sort network, star network, and hypercube [21], [22].
Although research has been conducted on node-disjoint paths of transposition networks [24], the literature on the construction of independent spanning trees for transposition networks is scant. In this article, we present a novel algorithm for constructing maximal independent spanning trees for transposition networks.

The remainder of this article is organized as follows. Section II provides definitions of transposition networks and introduces some necessary properties. Section III presents the proposed algorithm for constructing independent spanning trees for transposition networks. Section IV provides a demonstration of the validity of the proposed algorithm. Finally, the conclusion are provided in Section V.

II. PRELIMINARIES

This section provides definitions of transposition networks and some other relevant symbols.

Definition 1: Independent Spanning Trees constitute a set of spanning trees that are rooted on the same node r in a graph G such that any arbitrary node except r forms node-disjoint paths to the root.

Example 1: Consider, for example, Figure 1, according to the graph in Figure 1(a), we construct three spanning trees, as illustrated shown in Figure 1(b), Figure 1(c), and Figure 1(d). When we select a fixed root node \( r = 123 \) and any other arbitrary node \( v \), such as \( 321 \), the paths from \( v \) to \( r \) of the three spanning trees are as follows:

\[
\begin{align*}
  p_1 &= (321, 231, 213, 123) \\
  p_2 &= (321, 123) \\
  p_3 &= (321, 312, 132, 123)
\end{align*}
\]

We can observe that except for the selected node \( v = 321 \) and the root \( r = 123 \), each of the three paths passes through different edges. In addition to the selected node, choosing any other node except for the root would result in the formation of node-disjoint paths. Thus, the three spanning trees form a set of independent spanning trees.

Definition 2: An \( n \)-transposition network, denoted as \( T_n \), contains \( n! \) nodes. Each node is labeled with an unique address that belongs a 1, 2, \ldots, \( n \) set of permutations. A node \( v \) that belongs to a transposition network is adjacent to nodes whose addresses switch any two arbitrary digits with the digits in the address of \( v \).

Figure 1(a) and Figure 2 present examples of \( T_3 \) and \( T_4 \), respectively. Note that the networks can be presented in other forms than those in Figure 1(a) and Figure 2.

For a node \( v \), we define \( v_i \) as the ith digit of the node. Thus, \( v = v_1 v_2 \ldots v_n \). For the neighbor of \( v \), \( N(v) \) can be presented as follows:

\[
N(v) = v_1 \ldots v_{i-1} v_j v_{i+1} \ldots v_{j-1} v_j v_{j+1} \ldots v_n,
\]

for \( 1 \leq i < j \leq n \) (1)

Example 2: Consider, for example, a node \( v = 1234 \) in \( T_4 \); the neighbor of \( v \) can be calculated using equation 1:

\[
N(v) = \begin{cases} 
  2134, & \text{if } i = 1, j = 2; \\
  3214, & \text{if } i = 1, j = 3; \\
  4231, & \text{if } i = 1, j = 4; \\
  1324, & \text{if } i = 2, j = 3; \\
  1432, & \text{if } i = 2, j = 4; \\
  1243, & \text{if } i = 3, j = 4.
\end{cases}
\]

Through the preceding calculation, we obtain six neighbors of \( v \). According to definition 2, every node in an \( n \)-transposition network is labeled with a unique address of \( n \) digits. Every digit can switch places with any other \( n - 1 \) digits. Therefore, we can deduce the following property:

Property 1: [22] An \( n \)-transposition network has connectivity \( \frac{n(n-1)}{2} \). Through property 1, we can calculate the total edges of transposition network \( G \),

\[
E(G) = \frac{1}{2}(V(G) \times N(v)) = \frac{n(n-1)n!}{4}.
\]
Property 2: [22] An n-transposition network is a bipartite graph.

Example 3: Figure 1(a) displays an example of $T_3$. We can easily redraw the graph to the graph presented in Figure 3.

The bipartite graph is conducive to problem-solving.

Property 3: [21] An n-transposition network is vertex and edge transitive.

Through property 3, the automorphism of transposition networks simplifies constructions of independent spanning trees that we can select a fixed node as root.

III. CONSTRUCTING INDEPENDENT SPANNING TREES ON $T_n$

This section introduces the proposed algorithm for constructing independent spanning trees on $T_n$. According to property 3, every transposition network is vertex and edge transitive, we select the address $12 \ldots n$ as the common root of all independent spanning trees on $T_n$. For clarity, we set the root node as $r$, and for convenience, we write $v = 1234$ as representing setting 1234 as the address of node $v$.

According to property 1, an n-transposition network has a connectivity of $\frac{n(n-1)}{2}$, we deduce that exists $\frac{n(n-1)}{2}$ node-disjoint paths from a node to root. Therefore, we assume that $T_n$ comprises maximum independent spanning trees $\frac{n(n-1)}{2}$ because every path from a node to root in different independent spanning trees remains independent. To specify an independent spanning tree on $T_n$, we define $T_n(i)$ as the $i$th independent spanning tree on $T_n$, where $i \in \{1, 2, \ldots, \frac{n(n-1)}{2}\}$.

A. COMMON FUNCTIONS FOR CONSTRUCTION

For constructing independent spanning trees, we propose an algorithm, namely Transform, that can implement a transposition operation, with $v$ being the selected node, $p$ being the first selected position of the node, and $q$ being the second selected position of the node.

Algorithm 1 Transform ($v, p, q$)

\[
\begin{align*}
&\text{Algorithm 1 } \text{Transform} \ (v, p, q) \\
&v' := v \\
&v'_p := v_q \\
&v'_q := v_p \\
&\text{return } v'
\end{align*}
\]

Example 4: Let $v = 2134$. To switch two selected digits, namely 2 and 4, of node $v$ for the purpose of deriving a neighbor of node $v$, we can substitute $p$ and $q$ with the positions of the digits in the Transform algorithm, respectively. The position of digit 2 is 1 and the position of digit 4 is 4. Thus, $p = 1$ and $q = 4$, and we can obtain the address 4132.

To prevent confusion regarding whether a digit is meant as a literal integer or a position index, we propose an algorithm, namely Location, to determine the position of the digit as its solution, with $v$ being the selected node, $d$ being the selected digit, and $n$ being the dimension of transposition network $n$ in which $v$ is set:

Algorithm 2 Location ($v, d, n$)

\[
\begin{align*}
&i := 1 \\
&l := 1 \\
&\text{while } i \leq n \text{ do} \\
&\quad \text{if } v_i = d \text{ then} \\
&\quad \quad l := i \\
&\quad \quad i := n + 1 \\
&\quad i := i + 1 \\
&\text{return } l
\end{align*}
\]

To construct independent spanning trees, we must specify the neighbor selection procedure. The Select algorithm (Algorithm 3) can select different neighbors of a node based on a selected vertex $v$, the identification $i$ of the spanning tree, and the dimension of the transposition network $n$. The algorithm can ensure the selection of different neighbors with the same $v$ and $n$ for each identification $i$ of a spanning tree.

Algorithm 3 Select($v, i, n$)

\[
\begin{align*}
&p := 1 \\
&q := i + 1 \\
&\text{while } q > n \text{ do} \\
&\quad p := p + 1 \\
&\quad q := q-n+p \\
&\text{return Transform($v, p, q$)}
\end{align*}
\]

We subsequently present the construction of independent spanning trees on transposition networks of low dimension, and expand the dimension by induction. Because $T_1$ is a singleton graph and $T_2$ is a path graph on two vertices, the problem is trivial. Therefore, we start our construction on a three-dimensional transposition network. Before describing the algorithm, we introduce the following definitions, which are the foundations of our algorithms.

Definition 3: The function Create_Tree($r$) creates a graph with root node $r$.

Definition 4: Let $v$ be a node in a graph $G$. The function Add_Link($v, u$) adds a new node $u$ to $G$, with $u$ being connected with $v$.

Definition 5: Let $s$ be a vector of nodes in a graph $G$. The function Get_Element($s$) returns a node in the queue $s$. 

FIGURE 3. Example of $T_3$ in the form of a bipartite graph.
Algorithm 4  **Construct_T3**(r, i, n)

```plaintext
CREATE_TREE(r)
v := SELECT(r, i, n)
ADD_LINK(r, v)
s := N(v)

while s ≠ null do
    v' := GET_ELEMENT(s)
    if v' ≠ r then
        ADD_LINK(v, v')
        if i = 1 then
            ADD_LINK(v', TRANSFORM(v', 1, 2))
        REMOVE(s, v')
    if i ≠ 1 then
        s' := N(r)
        while s' ≠ null do
            v' := GET_ELEMENT(s')
            ADD_LINK(TRANSFORM(v, 1, 2), v')
        REMOVE(s', v')

return
```

We can classify the construction process executed using the algorithm into two cases.

**Case 1:** Select(r, i, n) = Transform(r, 1, 2)

Only one spanning tree is constructed in Case 1, as illustrated in Figure 4(a).

**Case 2:** Select(r, i, n) ≠ Transform(r, 1, 2)

Figure 4(b) and Figure 4(c) present examples of trees constructed in Case 2.

**Algorithm 5  **Parent_T3**(v, i, n)

```plaintext
p := null
u := SELECT(r, i, n)

if v ≠ r then
    if v = u then
        p := r
    else if v ∈ N(r) then
        if i = 1 then
            p := TRANSFORM(v, 1, 2)
        else
            p := TRANSFORM(u, 1, 2)
    else
        p := u

return p
```

Due to the top-down nature of the construction algorithm, whether the spanning trees constructed by the algorithm include all nodes may be unclear. Therefore, we propose an algorithm, namely Parent_T3 (Algorithm 5). Parent_T3 returns the parent of node v except for the root; it is based on the selected node v, identification of the spanning tree i, and the dimension of the transposition network n. Note that we always select r = 12...n as the root for consistency.

**C. CONSTRUCTING INDEPENDENT SPANNING TREES ON Tn**

Next, for T_n with n ≥ 4, the construction of independent spanning trees requires special technique to address the problem.

**Definition 7:** In T_n, an i-container, i = 1, 2, ..., n (n refers to the dimension of the network) represents a set that includes all the nodes for which the last digit equals i.

The container concept is essential for constructing independent spanning trees on T_n. Figure 5 shows an example.
According to the group method in definition 7, we define a concept about transposition position:

**Definition 8:** A fixed node in every container is adjacent to a node with the same transposition position, and the transposition position of a node is the node that takes a particular Transform operation on the fixed node.

In this article, we selected root and Transform(v, 1, 2) for v ∈ N(r) in any other container as fixed nodes.

**Example 5:** Taking T₄ as the example. Consider 4-container and 3-container, we selected root and Transform(v, 1, 2), namely v = 2143, as fixed nodes. As we implement Transform operation to both fixed nodes, such as Transform(v, 1, 3), we will receive v₁ = 3214 and v₂ = 4123 respectively. v₁ and v₂ is said taking same transposition position (see Figure 6).

Based on our algorithm, we can distinguish three cases:

**Case 1:** (Select(r, i, n)_i)ᵢ ≠ rᵢ; the children of the root do not belong to the n-container.

**Step 1** First, select a node that does not belong to the n-container as the only child c of the root (see Figure 7).

**Step 2** Next, select all neighbors of c in the current container as the children of c (see Figure 8).

**Step 3** Next, select all neighbors of Transform(c, 1, 2) in the current container as the children (see Figure 9).

**Step 4** Finally, select all neighbors of the nodes in the current container (see Figure 10).
Case 2: \( \text{Select}(r, i, n) \neq \text{Transform}(r, 1, 2) \) and \( (\text{Select}(r, i, n))_n = r_n \); the children of the root belong to the \( n \)-container, except for the node in Special Case.

Step 1 First, inside the \( n \)-container, construct a tree that exhibits the same behavior as that of \( T_3 \) Case 2 (see Figure 11). Note that the node \( v = \text{Transform}(r, 1, 2) \) belongs to the special case and is thus not selected.

Step 2 Next, select all neighbors of \( \text{Select}(r, i, n) \) as the children of \( \text{Select}(r, i, n) \) (see Figure 12).

Step 3 Finally, execute selection and construction processes similar to those in \( T_3 \) Case 2 (see Figure 13).

Special case: \( \text{Select}(r, i, n) = \text{Transform}(r, 1, 2) \); the children of the root are denoted \( \text{Transform}(r, 1, 2) \) and are treated as a special case.

Step 1 First, inside the \( n \)-container, construct a tree that exhibits same behavior that of \( T_3 \) Case 1 (see Figure 14).

Step 2 Next, for \( \text{Transform}(r, 1, 2) \) and \( N(\text{Transform}(r, 1, 2)) \) in the \( n \)-container, select all neighbors in any other container as their children (see Figure 15).

Step 3 Finally, for the nodes \( v \) selected in Step 2, select \( \text{Transform}(v, 1, 2) \) as the children (see Figure 16).

With the cases above, we design algorithm 6 presenting the constructions of independent spanning trees on \( T_n \) with \( n \geq 4 \), with \( v \) representing the selected node, \( i \) representing the identification of the spanning tree, and \( n \) representing the dimension of the transposition network.

Algorithm 7 shows the Parent function for \( n \geq 4 \), with \( v \) representing the selected node, \( i \) representing the identification of the spanning tree, and \( n \) representing the dimension of the transposition network. Note that we always retain the same root node for consistency.

Table 1 lists every node and its parent in every independent spanning tree.

Figures 17-19 show examples of Case 1, Case 2, and Special case pertaining to the construction of independent spanning trees for \( T_4 \) using our algorithm.
Algorithm 6  **Construct**\(_T_n\) \((r, i, n)\)

```plaintext
CREATE_TREE\(_r\)
\(v := \text{SELECT}(r, i, n)\)
ADD_LINK\(_r\), \(v\)
\(s := N(v)\setminus\{r\}\)

while \(s \neq \text{null} \) do
  \(v' := \text{GET_ELEMENT}(s)\)
  ADD_LINK\(_v\), \(v'\)
  \(s' := N(v')\setminus\{v\}\)
  if \(i = 1\) then
    if \(v_n = r_n\) then
      while \(s' \neq \text{null} \) do
        \(w := \text{GET_ELEMENT}(s')\)
        if \(w_n \neq v'_n\) then
          ADD_LINK\(_v\), \(w\)
          ADD_LINK\(_w\), \(\text{TRANSFORM}(w, 1, 2)\)
          REMOVE\(_s'\), \(w\)
        end if
      end while
    end if
  end if
  \(w := \text{GET_ELEMENT}(s')\)
  if \(v' = \text{TRANSFORM}(v', 1, 2)\) and \(w_n = r_n\) then
    ADD_LINK\(_v\), \(w\)
    if \(w_n = v'_n\) then
      ADD_LINK\(_w\), \(\text{TRANSFORM}(w, 1, 2)\)
      \(s'' := N(v'),\{v'\}\)
      while \(s'' \neq \text{null} \) do
        \(w' := \text{GET_ELEMENT}(s'')\)
        if \(w'_n = w_n\) then
          ADD_LINK\(_w\), \(w'\)
          REMOVE\(_s''\), \(w'\)
        end if
      end while
    end if
  end if
else if \(v_n = r_n\) then
  while \(s' \neq \text{null} \) do
    \(w := \text{GET_ELEMENT}(s')\)
    if \(v' = \text{TRANSFORM}(v', 1, 2)\) and \(w_n = r_n\) then
      ADD_LINK\(_v\), \(w\)
      if \(w_n = v'_n\) then
        ADD_LINK\(_w\), \(\text{TRANSFORM}(w, 1, 2)\)
        \(s'' := N(v'),\{v'\}\)
        while \(s'' \neq \text{null} \) do
          \(w' := \text{GET_ELEMENT}(s'')\)
          if \(w'_n = w_n\) then
            ADD_LINK\(_w\), \(w'\)
            REMOVE\(_s''\), \(w'\)
          end if
        end while
      end if
    end if
  end while
else
  while \(s' \neq \text{null} \) do
    \(w := \text{GET_ELEMENT}(s')\)
    if \(w_n \neq v'_n\) then
      ADD_LINK\(_v\), \(w\)
    end if
    \(s'' := N(w),\{v'\}\)
    while \(s'' \neq \text{null} \) do
      \(w' := \text{GET_ELEMENT}(s'')\)
      if \(w'_n = w_n\) then
        ADD_LINK\(_w\), \(w'\)
        REMOVE\(_s''\), \(w'\)
      end if
    end while
  end while
end if

return \(s, v'\)
```

Algorithm 7  **Parent** \((v, i, n)\)

```plaintext
\(p := \text{null}\)
\(u := \text{SELECT}(r, i, n)\)
if \(v \neq r\) then
  if \(v = u\) then
    \(p := r\)
  else if \(u_n = r_n\) then
    if \(v \in N(r)\) then
      if \(i = 1\) then
        \(p := \text{TRANSFORM}(v, 1, 2)\)
      else
        \(p := \text{TRANSFORM}(u, 1, 2)\)
      end if
    else
      \(p := u\)
    end if
  end if
else
  if \(v \in N(\text{SELECT}(r, i, n))\) then
    \(p := u\)
  else
    if \(i = 1\) then
      if \(v \in N(r)\) or \(N(v) \cap N(r) \neq \phi\) then
        \(p := \text{TRANSFORM}(v, 1, 2)\)
      else
        \(p := \text{TRANSFORM}(u, \text{LOCATION}(v, 4, n), 4)\)
      end if
    else
      \(w := \text{TRANSFORM}(u, \text{LOCATION}(u, v_n, n), 4)\)
      if \(N(v) \cap N(r) \neq \phi\) then
        \(p := \text{TRANSFORM}(w, 1, 2)\)
      else
        \(p := w\)
      end if
    end if
  end if
else
  if \(v \in N(u)\) then
    \(p := u\)
  else if \(v_n = u_n\) then
    \(p := \text{TRANSFORM}(u, 1, 2)\)
  else
    \(p := \text{TRANSFORM}(v, \text{LOCATION}(v, u_n, n), 4)\)
  end if
end if

return \(p\)
```

**IV. PROOF OF VALIDITY**

This section demonstrates the validity of the algorithm. Because of the few nodes of \(T_3\), we can check the independence of spanning trees directly. We start our proof with \(n \geq 4\).

**Lemma 1:** Every pair of nodes in any two different containers have no share neighbors.

**Proof:** Because the last digits of the addresses of the nodes inside a container are the same, every node remains unique as we hide its last digit. To move to another container, the last digit of the address must be changed because it is not equal to the last digit in the container at the destination. Therefore, we search the target container and make appropriate exchanges for the final digits. Because all nodes in a container are unique, the addresses of the nodes remain unique when we exchange the last digits. \(\square\)

**Theorem 1:** For \(n \geq 4\), \(T_{n-1}\) is a part of \(T_n\).

**Proof:** Consider the container problem; we can obtain all nodes of \(T_3\) in a 4-container. The relation between nodes remains the same as we hide the last digits of the addresses (neighbors are still in relation of transposition). Therefore, \(T_3\) is a part of \(T_4\). We can obtain all nodes of \(T_{n-1}\) in any \(i\)-container, \(i = 1, 2, 3, \ldots, n\). A transposition operation from a node to another node in the same container never switch with the last digit. We can consider it as an \(n - 1\)-transposition
network because relation between nodes remains transposition with ignoring the last digits. Therefore, for \( n \geq 4 \), \( T_{n-1} \) is a part of \( T_n \).

**Lemma 2:** All nodes in a container and all neighbors of them includes all nodes of the transposition network.

**Proof:** In a container, all nodes are unique. Because a node is adjacent to \( \frac{n(n-1)}{2} \) nodes, the number of neighbors...
inside the container can be subtracted; a node can connect to \( \frac{n(n-1)}{2} - \frac{(n-1)(n-2)}{2} = n - 1 \) nodes outside the current container. According to Lemma 1, we can conclude that for all the nodes in a container, each node does not share neighbors in any other container, which means all the neighbors are unique. Because a container contains \((n-1)!\) nodes, we can directly multiply it: \((n-1)! \times (n-1)! = (n-1)(n-1)! \) nodes. After adding back the number of nodes in the container, we can obtain the number of all nodes in a transposition network.

**Theorem 2:** For \( n \geq 4 \), the independent spanning trees of \( T_{n-1} \) are a part of the independent spanning trees of \( T_n \).

**Proof:** On the basis of Algorithm 7, we implement a version of algorithm 4 designed for \( T_3 \) inside a container. Because Algorithm 4 forms independent spanning trees, we can conclude that the independent spanning trees of \( T_3 \) are a part of the independent spanning trees of \( T_4 \) (see Figure 20). We can expand the proof by enlarging the bipartite set of a container for higher dimensions. In a container for every container base construction, the algorithm always do an \( T_3 \)-like construction. The spanning trees of \( T_n \) are clearly expansion of \( T_{n-1} \) by connecting nodes to central nodes. Because the basic algorithm remains the same, we can conclude that for \( n \geq 4 \), the independent spanning trees of \( T_{n-1} \) are a part of the independent spanning trees of \( T_n \).

**Theorem 3:** For \( n \geq 4 \), \( T_n(1), T_n(2), \ldots, T_n\left(\frac{n(n-1)}{2}\right) \) are \( \frac{n(n-1)}{2} \) independent spanning trees of \( T_n \) constructed by the \texttt{Construct}_4 \( r, i, n \) algorithm.

**Proof:** We prove the correctness of our algorithm with the following cases.

**Case 1:** The constructions of Case 1 are base on a single container except root container. Consider Figures 7-10. Through the mentioned algorithm \texttt{Construct}_4 \( (r, i, n) \), we can walk through a container first. Note that the node first walked in has the same transposition position as \texttt{Transform}(\( r, 1, 2 \)), which belongs to Special case. Subsequently, we walk through the container with \( T_3 \)-like behavior. Finally we walk through all neighbors of the nodes that belong to this container. According to Lemma 2, we can ensure that we walk through all nodes of transposition network. Because each container is isolated from other containers, we can conclude that the spanning trees are independent of one another. Through our algorithm, we can construct \( n-1 \) independent spanning trees in Case 1.

**Case 2:** The construction of Case 2 are base on multiple containers. Consider Figures 11-13. Through the algorithm \texttt{Construct}_4 \( (r, i, n) \), we can walk through the starting container without the way in Special case. Compared with Case 1, the construction is valid because every node created in Case 1 in root container is a leaf node. Next, we walk through a node in every container. Consider Figures 11-13. Through the algorithm \texttt{Construct}_4 \( (r, i, n) \), we can walk through the starting container without the way in Special case. Compared with Case 1, the construction is valid because every node created in Case 1 in root container is a leaf node. Next, we walk through all neighbors of \texttt{Transform}(\( r, 1, 2 \)) and \texttt{Transform}(\( N(r), 1, 2 \)). Compared with Case 2, when we construct a node-disjoint path back to the root, we can walk through the neighbor of the root and then back to the root in Case 2. By contrast, in Special case, we can walk through the neighbor of the aforementioned node, and can walk through \texttt{Transform}(\( r, 1, 2 \)).
nodes and directed edges. The number of nodes in a transposition network is the summation of the numbers of nodes traversed once, the time complexity of the algorithm in an n-transposition network is $O(n^2 \times N)$, where $N$ is the number of nodes of an n-transposition network.

Proof: Because every node and every directed edge are traversed once, the time complexity of the algorithm in an n-transposition network is the summation of the numbers of nodes and directed edges. The number of nodes is $n!$ and the number of directed edges is $n! \times \frac{n(n-1)}{2}$. The time complexity is $O(n! + n! \times \frac{n(n-1)}{2}) = O(n! \times (1 + \frac{n(n-1)}{2})) = O(n^2 \times n!)$.

**V. CONCLUSION**

In this article, we study the problem of searching and constructing independent spanning trees on transposition networks, which are vital for interconnection networks. The present study is the first to construct maximal independent spanning trees on transposition networks.

Appealing topics for future work can involve determining low-cost methods for constructing independent spanning trees. It is a challenge to pursue the goal of reducing the heights of ISTs (the longest path from the root to any leaf) and the time complexity of the algorithm. Moreover, many problems involving Cayley graphs and other interconnection networks remain unsolved. We expect that our algorithm can make notable contributions to graph theory.

**REFERENCES**

[1] S. B. Akers and B. Krishnamurthy, “A group-theoretic model for symmetric interconnection networks,” IEEE Trans. Comput., vol. 38, no. 4, pp. 555–566, Apr. 1989.
[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] 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.
[4] 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.
[5] 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.
[6] P. J. Chase, “Transformation graphs,” SIAM J. Comput., vol. 2, no. 2, pp. 128–133, Jun. 1973.
[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. L. Cheng, J. X. Fan, X. H. Jia, S. K. Zhang, and B. R. Chen, “Constructive algorithm of independent spanning trees on Möbius cubes,” Comput. J., vol. 56, no. 11, pp. 1347–1362, Nov. 2013.
[9] D. Clark, “Transposition graphs: An intuitive approach to the parity theorem for permutations,” Math. Mag., vol. 78, no. 2, pp. 124–130, Apr. 2005.
[10] Y.-Q. Feng, “Automorphism groups of Cayley graphs on symmetric groups with generating transposition sets,” J. Combinat. Theory, B, vol. 96, no. 1, pp. 67–72, Jan. 2006.
[11] A. Ganesan, “Automorphism group of the complete transposition graph,” J. Algebraic Combinatorics, vol. 42, no. 3, pp. 793–801, Nov. 2015.
[12] S.-Y. Hsieh and C.-J. Tu, “Constructing edge-disjoint spanning trees in locally twisted cubes,” Theor. Comput. Sci., vol. 410, nos. 8–10, pp. 926–932, Mar. 2009.
[13] Z. Hussain, B. ALBdaiwi, and A. Cerny, “Node-independent spanning trees in Gaussian networks,” J. Parallel Distrib. Comput., vol. 109, pp. 324–332, Nov. 2017.
[14] 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.
[15] Y. Iwasaki, Y. Kajiwara, K. Obokata, and Y. Igarashi, “Independent spanning trees of chordal rings,” Inf. Process. Lett., vol. 69, no. 3, pp. 155–160, Feb. 1999.
[16] J. S. Jwo, “Properties of star graph, bubble-sort graph, Perfax-reversal graph and complete-transposition graph,” J. Inf. Sci. Eng., vol. 12, no. 4, pp. 603–617, 1996.
[17] 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. Comput., Switzerland: Springer, Dec. 2017, pp. 41–55.
[18] S. S. Kao, I. M. Chang, K. J. Pai, and R. Y. Wu, “Constructing independent spanning trees on bubble-sort networks,” in Proc. Int. Conf. Combinatorics Conf. Cham, Switzerland: Springer, Jul. 2018, pp. 1–13.
[19] 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.
[20] S. Khuller and B. Schieber, “On independent spanning trees,” Inf. Process. Lett., vol. 42, no. 6, pp. 321–323, 1992.
[21] S. Lakshminarahan, J.-S. Jwo, and S. K. Dhall, “Symmetry in interconnection networks based on Cayley graphs of permutation groups: A survey,” Parallel Comput., vol. 19, no. 4, pp. 361–407, Apr. 1993.
[22] S. Latifi and P. K. Srimani, “Transposition networks as a class of fault-tolerant robust networks,” IEEE Trans. Comput., vol. 45, no. 2, pp. 230–238, Feb. 1996.
[23] 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.
[24] Y. Suzuki, K. Kaneko, and M. Nakamori, “Node-disjoint paths algorithm in a transposition graph,” IEICE Trans. Inf. Syst., vol. 89, no. 10, pp. 2600–2605, Oct. 2006.
[25] R. Walker, “Implementing discrete mathematics: Combinatorics and graph theory with Mathematica,” in The Math. Gazette, vol. 76, no. 476, S. Skiena Ed. Reading, MA, USA: Addison-Wesley, 1992, pp. 286–288.
[26] 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.

[27] Y. Wang, J. Fan, G. Zhou, and X. Jia, “Independent spanning trees on twisted cubes,” J. Parallel Distrib. Comput., vol. 72, no. 1, pp. 58–69, Jan. 2012.

[28] 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.

[29] M.-Y. Xu, “Automorphism groups and isomorphisms of Cayley digraphs,” Discrete Math., vol. 182, nos. 1–3, pp. 309–319, Mar. 1998.

[30] 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.

[31] J.-S. Yang, J.-M. Chang, S.-M. Tang, and Y.-L. Wang, “On the independent spanning trees of recursive circulant graphs G(cd, d) with d > 2,” Theor. Comput. Sci., vol. 410, nos. 21–23, pp. 2001–2010, 2009.

[32] 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.

CHIEN-FU LIN received the B.S. degree from the Department of Computer Science and Information Engineering, National Chung Cheng University, Taiwan, in 2014. He is currently pursuing the master’s degree with the Department of Computer Science and Information Engineering, National Cheng Kung University, Taiwan.

JIE-FU HUANG received the B.S. degree from the Information Management Department, National Taiwan University, Taipei, Taiwan, in June 2003, and the M.S. degree from the Information Management Institute, National Cheng Kung University, Tainan, Taiwan, in June 2005. He is currently pursuing the Ph.D. degree with the Department of Computer Science and Information Engineering, National Cheng Kung University. His current research interests include design and analysis of algorithms and graph theory.

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. He also joins 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.

* * *