Parallel construction of optimal independent spanning trees on hypercubes

Jinn-Shyong Yang a,d, Shyue-Ming Tang b, Jou-Ming Chang a, Yue-Li Wang c,d,*

a Department of Information Management, National Taipei College of Business, Taipei, Taiwan, ROC
b Department of Psychology, National Defense University, Taipei, Taiwan, ROC
c Department of Computer Science and Information Engineering, National Chi Nan University, University Road, Puli, Nantou Hsien 545, Taiwan, ROC
d Department of Information Management, National Taiwan University of Science and Technology, Taipei, Taiwan, ROC

Received 22 August 2006; received in revised form 10 November 2006; accepted 13 December 2006
Available online 21 December 2006

Abstract

The use of multiple independent spanning trees (ISTs) for data broadcasting in networks provides a number of advantages, including the increase of fault-tolerance and bandwidth. Thus, the designs of multiple ISTs on several classes of networks have been widely investigated. Tang et al. [S.-M. Tang, Y.-L. Wang, Y.-H. Leu, Optimal independent spanning trees on hypercubes, Journal of Information Science and Engineering 20 (2004) 143–155] studied the problem of constructing $k$ ISTs on $k$-dimensional hypercube $Q_k$, and provided a recursive algorithm for their construction (i.e., for constructing $k$ ISTs of $Q_k$, it needs to build $k - 1$ ISTs of $Q_{k-1}$ in advance). This kind of construction forbids the possibility that the algorithm could be parallelized. In this paper, based on a simple concept called Hamming distance Latin square, we design a new algorithm for generating $k$ ISTs of $Q_k$. The newly proposed algorithm relies on a simple rule and is easy to be parallelized. As a result, we show that the ISTs we constructed are optimal in the sense that both the heights and the average path length of trees are minimized.

© 2006 Elsevier B.V. All rights reserved.

Keywords: Independent spanning trees; Hypercubes; Internally disjoint paths; Latin square; Hamming distance; Algorithms

1. Introduction

The $k$-dimensional hypercube, denoted by $Q_k$, is a graph consists of $N = 2^k$ vertices represented by binary strings of length $k$, and two vertices are adjacent whenever their corresponding strings differ in exactly one place. Alternately, hypercubes can also be defined recursively using the operations of cartesian product,
i.e., $Q_k = Q_{k-1} \times K_2$ and $Q_1 = K_2$ is a complete graph with two vertices. By the recursive definition, we see that the regularity and connectivity of $Q_k$ are persisted in accord with $k$. For example, $Q_4$ is shown in Fig. 1.

Hypercubes play an important role in parallel computing systems because of their simple structure and suitability for developing algorithms. For hypercubes, a large amount of literatures have been devoted to the research on their topological properties as well as on applications of parallel computing (e.g., see [6] for a comprehensive survey paper or [14] for a monograph).

The vertex set and the edge set of a graph $G$ are denoted by $V(G)$ and $E(G)$, respectively. Two paths $P$ and $Q$ connecting a vertex $x$ to a vertex $y$ are said to be internally disjoint, denoted by $P \parallel Q$, if $E(P) \cap E(Q) = \emptyset$ and $V(P) \cap V(Q) = \{x, y\}$. A tree $T$ is called a spanning tree of a graph $G$ if $V(T) = V(G)$. Further, $T$ is a rooted spanning tree if it provides a specified vertex called the root of $T$. If $T$ is a tree and $x, y \in V(T)$, we denote $T[x, y]$ as the unique path from $x$ to $y$ in $T$. Two spanning trees $T$ and $T'$ of a graph $G$ are said to be independent (refer as ISTs for short) if they are rooted at the same vertex, say $r$, and such that $T[r, x] \parallel T'[r, x]$ for every vertex $x \in V(G) \setminus \{r\}$. Also, we refer a set of spanning trees of $G$ to be independent if they are pairwise independent. For example, four ISTs rooted at vertex 0 for $Q_4$ are shown in Fig. 2.

The design of multiple ISTs has applications to the reliable communication protocols [1,10]. For example, a rooted spanning tree in the underlying graph of a network can be viewed as a broadcasting scheme for data communication. Thus, the fault tolerance can be achieved by sending $k$ copies of the message along the $k$ ISTs rooted at the source node. Recently, the problem of constructing multiple ISTs of a given graph has received much attention. However, this is a very hard problem for arbitrary graphs. In fact, Zehavi and Itai [22] conjectured that for any $k$-connected graph $G$ and each vertex $r$ of $G$, there exist $k$ ISTs of $G$ rooted at $r$. The conjecture has been confirmed only for $k$-connected graphs with $k \leq 4$ in [2,4,10,22], and it is still open for arbitrary $k$-connected graphs when $k \geq 5$. Moreover, by providing the construction schemes of ISTs, the conjecture has been proved to hold for several restricted classes of graphs or digraphs (especially, the graph classes related to interconnection networks), such as planar graphs [8,9,15,16], product graphs [17], chordal rings [11,21], deBruijn and Kautz digraphs [5,7], and hypercubes [20] etc. Note that the development of algorithms for constructing ISTs tends toward pursuing two research goals, one is the design of efficient construction schemes (e.g., see ISTs for short) and another is the reducing the heights of ISTs (e.g., see [7,20,21] for height improvements).

As to the construction of the height-reduced ISTs on hypercubes, Tang et al. [20] proposed an algorithm that is modified from the algorithm for product graphs in [17] and showed that the resulting ISTs produced from their algorithm is optimal in the sense of average path length, where the path length of a tree is defined to be the sum of the distance from every vertex to the root in the tree. Due to the definition of hypercubes using cartesian product, both algorithms in [17,20] are designed by a recursive fashion in a natural way (i.e., for constructing $k$ ISTs of $Q_k$, we need to build $k - 1$ ISTs of $Q_{k-1}$ in advance). This kind of construction forbids the possibility that the algorithm could be parallelized. In this paper, we would like to give a new algorithm to generate $k$ ISTs of $Q_k$. Our generation is based on a concept called Hamming distance Latin square (see Section 2 for definition) which is inspired from the idea of [3,13] for solving the parallel routing problem on

![Fig. 1. A 4-dimensional hypercube $Q_4$.](image-url)
particular networks. As a result, we will show that our generation relies on a simple rule and it can easily be parallelized (i.e., \( k \) ISTs are established simultaneously). Moreover, all the \( k \) ISTs obtained from our algorithm are isomorphic if we disregard the representation of binary strings of vertices (e.g., see Fig. 2).

The remaining part of this paper is organized as follows. Section 2 introduces the concept of Hamming distance Latin square and gives some useful properties. Section 3 presents our algorithm to generate \( k \) ISTs on hypercube \( Q_k \) and shows the correctness. Finally, a concluding remark to the discussion about the height and the average path length of ISTs are given in the last section.

2. Preliminaries

Before presenting our algorithm, we first give some notations and useful properties. The neighborhood of a vertex \( x \) in a graph \( G \), denoted by \( N_G(x) \), is the set of vertices adjacent to \( x \) in \( G \). For a tree \( T \) rooted at a vertex \( r \), the parent of a vertex \( x \neq r \), denoted by parent \((T, x)\), is a vertex adjacent to \( x \) in the path \( T[r, x] \).

Since hypercubes are vertex-symmetric, without loss of generality, we may consider vertex 0 as the root of ISTs for \( Q_k \). Suppose \( \mathcal{F} \) is a set of \( k \) ISTs rooted at 0 for \( Q_k \). Obviously, \( N_{Q_k}(0) = \{2^0, 2^1, \ldots, 2^{k-1}\} \). From the connectivity of \( Q_k \) and the definition for \( \mathcal{F} \), there is only one child of 0 in every spanning tree \( T \in \mathcal{F} \). Thus, if the root takes \( 2^i (0 \leq i \leq k - 1) \) as its child, we denote by \( T_i \) for the specified spanning tree. For example, four ISTs of \( Q_4 \) shown in Fig. 2 are denoted by \( T_0, T_1, T_2, \) and \( T_3 \) from left to right, respectively.

To transmit multiple data items from the source node to the destination node on \( Q_k \) simultaneously, there are a few algorithms that allow us to adopt \( k \) disjoint paths. To our knowledge, the previously existing algorithms may be found in [3,12,18,19]. Particularly, a special matrix called Hamiltonian circuit Latin square (HCLS) was used in [3] to find a set of vertex-disjoint paths on hypercubes (see also [13] for the analogue on recursive circulant networks). In the following, a concept borrowed from HCLS, which we refer to as Hamming distance Latin square, will be introduced. The following are the relevant definitions.

A Latin square is a square matrix with \( n^2 \) entries chosen from a set \( H \) of \( n \) distinct elements such that none of the elements occurs twice within any row or column of the matrix. For the hypercube \( Q_k \) and a vertex \( x(\neq 0) \in Q_k \) with binary string \( x = x_{k-1}x_{k-2} \cdots x_0 \), a Hamming distance Latin square (HDLS for short) with respect to \( x \) is a Latin square whose entries are chosen from the set \( H_x = \{i:0 \leq i \leq k-1 \text{ and } x_i = 1\} \). Note that the Hamming distance from \( x \) to 0 in \( Q_k \) is defined to be the value \( |H_x| \). In particular, we define the following two matrices.

**Definition 1.** Suppose that a vertex \( x(\neq 0) \) has Hamming distance \( t \) to the vertex 0 in \( Q_k \) and let \( H_x = \{i_0, i_1, \ldots, i_{t-1}\} \) such that \( i_0 < i_1 < \cdots < i_{t-1} \). Then, the matrices defined below are called increasing rotational HDLS and decreasing rotational HDLS, respectively, with respect to \( x \) in \( Q_k \):

\[
I_t(x) = \begin{bmatrix}
i_0 & i_1 & \cdots & i_{t-2} & i_{t-1} \\
i_1 & i_2 & \cdots & i_{t-1} & i_0 \\
\vdots & \vdots & \ddots & \vdots & \vdots \\
i_{t-2} & i_{t-1} & \cdots & i_1 & i_0 \\
i_{t-1} & i_0 & \cdots & i_{t-3} & i_{t-2}
\end{bmatrix}
\quad \text{and} \quad
D_t(x) = \begin{bmatrix}
i_{t-1} & i_{t-2} & \cdots & i_1 & i_0 \\
i_{t-2} & i_{t-3} & \cdots & i_0 & i_{t-1} \\
\vdots & \vdots & \ddots & \vdots & \vdots \\
i_1 & i_0 & \cdots & i_3 & i_2 \\
i_0 & i_{t-1} & \cdots & i_2 & i_1
\end{bmatrix}.
\]
In addition, the entry \(i_{j+1}\) in \(I_k(x)\) (respectively, \(i_{j-1}\) in \(D_k(x)\)) is called the successor of \(i_j\) and is denoted by \(\text{succ}(i_j)\), where the indices \(j\) and \(j + 1\) (respectively, \(j - 1\)) are taken modulo \(t\).

Throughout the rest we use \(I_k(x)\) as a demonstration. Indeed, by symmetry, we can refer to \(D_k(x)\) instead of \(I_k(x)\) to draw the same conclusion. Let \(A = a_1,a_2,\ldots,a_t\) be a sequence of nonnegative integers. A prefix-powersum (with base 2) of \(A\) is defined to be the sum \(\sum_{i=1}^{t} 2^{a_i}\) for some \(1 \leq l \leq t\). In particular, a prefix-powersum is strict if \(l \neq t\). Because each row in the matrix \(I_k(x)\) can be viewed as a nonnegative integer sequence, we are easy to verify the following property.

**Proposition 2.1.** Let \(p\) and \(q\) be any two distinct rows of \(I_k(x)\). If \(s_1\) is a strict prefix-powersum of row \(p\) and \(s_2\) is a strict prefix-powersum of row \(q\), then \(s_1 \neq s_2\).

Since every positive integer can be decomposed into the sum of powers of 2, we say the decomposition of an integer to mean the terms for those powers.

**Proposition 2.2.** Let \(x(\neq 0) \in Q_k\) be a vertex with binary string \(x = x_{k-1}x_{k-2}\cdots x_0\) and suppose that \(x_i = 0\) for some \(i\) with \(0 \leq i \leq k - 1\). Let \(y = x + 2^l\) (i.e., \(y\) is obtained from \(x\) by setting \(x_i = 1\)) and suppose that a row in \(I_k(y)\), say \(p\), contains \(i\) as the last element. If \(s_1\) is a strict prefix-powersum of \(p\) in \(I_k(y)\), then \(x - s_1 \neq y - s_2\).

**Proof.** It follows from the fact that the decomposition of \(x - s_1\) does not include the term \(2^l\), but \(y - s_2\) contains the term \(2^l\).

**Proposition 2.3.** Let \(x(\neq 0) \in Q_k\) be a vertex with binary string \(x = x_{k-1}x_{k-2}\cdots x_0\) and suppose that \(x_i = x_j = 0\) for some \(i, j\) with \(0 \leq i, j \leq k - 1\) and \(i \neq j\). Let \(y = x + 2^l\) and \(z = x + 2^j\). Suppose that the row \(p\) in \(I_k(y)\) contains \(i\) as the last element and the row \(q\) in \(I_k(z)\) contains \(j\) as the last element. If \(s_1\) is a strict prefix-powersum of \(p\) in \(I_k(y)\) and \(s_2\) is a strict prefix-powersum of \(q\) in \(I_k(z)\), then \(y - s_1 \neq z - s_2\).

**Proof.** Obviously, the decomposition of \(y - s_1\) contains the term \(2^l\), but does not include the term \(2^j\). Oppositely, \(z - s_2\) receives the contrary.

**Example 1.** We consider the vertex \(x = 1101(13)\) in \(Q_k\). The two particular matrices defined above are given by

\[
I_k(x) = \begin{bmatrix}
0 & 2 & 3 \\
2 & 3 & 0 \\
3 & 0 & 2
\end{bmatrix}
\quad \text{and} \quad
D_k(x) = \begin{bmatrix}
3 & 2 & 0 \\
2 & 0 & 3 \\
0 & 3 & 2
\end{bmatrix}.
\]

The strict prefix-powersums for each row in \(I_k(x)\) are the following: \(2^0 = 1\) and \(2^0 + 2^2 = 5\) for row 1; \(2^2 = 4\) and \(2^2 + 2^3 = 12\) for row 2; and \(2^3 = 8\) and \(2^3 + 2^0 = 9\) for row 3. Also, if we consider \(y = 1111(15)\), then

\[
I_k(y) = \begin{bmatrix}
0 & 1 & 2 & 3 \\
1 & 2 & 3 & 0 \\
2 & 3 & 0 & 1 \\
3 & 0 & 1 & 2
\end{bmatrix}.
\]

In this case, the entry 1 is the last element in the third row of \(I_k(y)\). The strict prefix-powersums for this row are \(2^2 = 4, 2^2 + 2^3 = 12,\) and \(2^3 + 2^0 = 9\). Thus, \(13 - s_1 \neq 15 - s_2\), where \(s_1 \in \{1,5,4,12,8,9\}\) and \(s_2 \in \{4,12,13\}\).

3. Constructing independent spanning trees on \(Q_k\)

To represent a path of \(Q_k\), we use the notation \(x \rightarrow y^\text{\(2^l\)}\) if \(x + 2^l = y\) (or \(x \rightarrow y\) if \(x - 2^l = y\)) to mean that both \(x\) and \(y\) are adjacent in the path. We now design an algorithm to construct a set \(\mathcal{T} = \{T_i : 0 \leq i \leq k - 1\}\) of ISTs rooted at 0 for \(Q_k\), where the construction can be carried out by describing the parent of every vertex in each spanning tree \(T_i\).
Algorithm \textbf{GEN-PARENTS}
begin
for every vertex \(x(\neq0) \in Q_k\) with binary string \(x = x_{k-1}x_{k-2} \cdots x_0\) do
  for \(i = 0\) to \(k - 1\) do
    if \((x_i = 1)\) parent\((T_i,x) = x - 2^{\text{succ}(i)}\); // where succ\((i)\) is referred to \(I_k(x)\)
    if \((x_i = 0)\) parent\((T_i,x) = x + 2^i\);
  enddo
enddo
end \textbf{GEN-PARENTS}

\textbf{Example 2.} We consider \(T_2\) constructed from \(Q_4\) as follows. Let \(x(\neq0) \in Q_4\) be any vertex with binary string \(x = x_3x_2x_1x_0\). Clearly, if \(x \in \{1,2,3,8,9,10,11\}\) then \(x_2 = 0\), and thus we have \(\text{parent}(T_2,x) = x + 2^2 = x + 4\) in this case. The remaining cases are the following: if \(x \in \{5,7\}\), then \(\text{succ}(2) = 0\) and \(\text{parent}(T_2,x) = x - 2^0 = x - 1\); if \(x = 4\), then \(\text{succ}(2) = 2\) and \(\text{parent}(T_2,4) = 4 - 2^2 = 0\); if \(x = 6\), then \(\text{succ}(2) = 1\) and \(\text{parent}(T_2,6) = 6 - 2^1 = 4\). Finally, if \(x \in \{12,13,14,15\}\), then \(\text{succ}(2) = 3\) and \(\text{parent}(T_2,x) = x - 2^3 = x - 8\). See \textbf{Table 1} for the complete informations about the construction of \(T_2\).

In what follows, we show the correctness of the algorithm.

\textbf{Lemma 3.1.} For each \(i = 0, 1, \ldots, k - 1\), \(T_i\) is a spanning tree rooted at 0 of \(Q_k\).

\textbf{Proof.} According to the algorithm, it is obvious that every vertex \(x(\neq0) \in Q_k\) implies \(x \in T_i\). In particular, the vertex \(2^i\) is adjacent to 0 in \(T_i\) since \(\text{succ}(i) = i\) in \(I_k(2^i)\). Since the two end vertices of any edge in \(T_i\) are with Hamming distance 1, \(T_i\) is a spanning subgraph of \(Q_k\). To complete the proof, we need to show that there exists a unique path from every vertex \(x(\neq0)\) to 0 in \(T_i\). Recall that \(H_x = \{i:0 \leq i \leq k - 1\}\) and \(x_i = 1\). We suppose that \(H_x = \{i_0,i_1,\ldots,i_{t-1}\}\) with \(i_0 < i_1 < \cdots < i_{t-1}\) and consider the following two cases:

\textbf{Case 1.} \(i \in H_x\) (i.e., \(x_i = 1\)). Assume \(i = i_p\) for some \(p = 0, 1, \ldots, t - 1\). By the rule, \(x\) is adjacent to \(x - 2^{p+1}\) in \(T_i\). Let \(y = x - 2^{p+1}\). Since \(H_y = \{i_0,i_1,\ldots,i_p,i_{p+2},\ldots,i_{t-1}\}\), \(y\) is adjacent to \(y - 2^{p+2}\) in \(T_i\). Let \(z = y - 2^{p+2}\). Clearly, \(H_z = \{i_0,i_1,\ldots,i_p,i_{p+3},\ldots,i_{t-1}\}\). Again by the rule, \(z\) is adjacent to \(z - 2^{p+3}\) in \(T_i\). By this way, we can find the following unique path connecting \(x\) and 0 in \(T_i\):
\[
x - 2^{p+1} \rightarrow (x - 2^{p+1}) - 2^{p+2} \rightarrow (x - 2^{p+1} - 2^{p+2}) - 2^{p+3} \rightarrow \cdots - 2^{p+1} \rightarrow (2^p) - 2^p \rightarrow 0.
\]

\textbf{Table 1}
The parent of a vertex \(x(\neq0) \in Q_k\) in \(T_2\)

<table>
<thead>
<tr>
<th>(x)</th>
<th>Binary string</th>
<th>(\text{succ}(i))</th>
<th>(2^{\text{succ}(i)})</th>
<th>(\text{parent}(T_2,x))</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0001</td>
<td>–</td>
<td>–</td>
<td>5</td>
</tr>
<tr>
<td>2</td>
<td>0010</td>
<td>–</td>
<td>–</td>
<td>6</td>
</tr>
<tr>
<td>3</td>
<td>0011</td>
<td>–</td>
<td>–</td>
<td>7</td>
</tr>
<tr>
<td>4</td>
<td>0100</td>
<td>2</td>
<td>4</td>
<td>0</td>
</tr>
<tr>
<td>5</td>
<td>0101</td>
<td>0</td>
<td>1</td>
<td>4</td>
</tr>
<tr>
<td>6</td>
<td>0110</td>
<td>1</td>
<td>2</td>
<td>4</td>
</tr>
<tr>
<td>7</td>
<td>0111</td>
<td>0</td>
<td>1</td>
<td>6</td>
</tr>
<tr>
<td>8</td>
<td>1000</td>
<td>–</td>
<td>–</td>
<td>12</td>
</tr>
<tr>
<td>9</td>
<td>1001</td>
<td>–</td>
<td>–</td>
<td>13</td>
</tr>
<tr>
<td>10</td>
<td>1010</td>
<td>–</td>
<td>–</td>
<td>14</td>
</tr>
<tr>
<td>11</td>
<td>1011</td>
<td>–</td>
<td>–</td>
<td>15</td>
</tr>
<tr>
<td>12</td>
<td>1100</td>
<td>3</td>
<td>8</td>
<td>4</td>
</tr>
<tr>
<td>13</td>
<td>1101</td>
<td>3</td>
<td>8</td>
<td>5</td>
</tr>
<tr>
<td>14</td>
<td>1110</td>
<td>3</td>
<td>8</td>
<td>6</td>
</tr>
<tr>
<td>15</td>
<td>1111</td>
<td>3</td>
<td>8</td>
<td>7</td>
</tr>
</tbody>
</table>
Case 2. \(i \not\in H_c\) (i.e., \(x_i = 0\)). By the rule, \(x \) is adjacent to \(x + 2^i\) in \(T_i\). Let \(y = x + 2^i\). Then \(i \in H_c\). From Case 1, we have shown that there exists a unique path, say \(P\), connecting \(y\) and 0 in \(T_i\). Therefore, \(x \xrightarrow{2^i} y\) along with the path \(P\) forms the unique path connecting \(x\) and 0 in \(T_i\). \(\square\)

Example 3. We consider 1101(13) and 0110(6) in \(Q_4\). From the above lemma, we can easily check the unique path from 13 (respectively, 6) to 0 in every spanning tree \(T_i\), as follows:

\[
T_0[13, 0] : 13 \xrightarrow{2^3} 9 \xrightarrow{2^3} 3 \xrightarrow{2^0} 0 \\
T_0[6, 0] : 6 \xrightarrow{2^0} 7 \xrightarrow{2^1} 5 \xrightarrow{2^1} 1 \xrightarrow{2^0} 0 \\
T_1[13, 0] : 13 \xrightarrow{2^3} 15 \xrightarrow{2^2} 11 \xrightarrow{2^3} 3 \xrightarrow{2^0} 2 \xrightarrow{2^1} 0 \\
T_1[6, 0] : 6 \xrightarrow{2^2} 2 \xrightarrow{2^1} 0 \\
T_2[13, 0] : 13 \xrightarrow{2^3} 5 \xrightarrow{2^0} 4 \xrightarrow{2^0} 0 \\
T_2[6, 0] : 6 \xrightarrow{2^1} 4 \xrightarrow{2^0} 0 \\
T_3[13, 0] : 13 \xrightarrow{2^3} 12 \xrightarrow{2^2} 8 \xrightarrow{2^3} 0 \\
T_3[6, 0] : 6 \xrightarrow{2^3} 14 \xrightarrow{2^1} 12 \xrightarrow{2^2} 8 \xrightarrow{2^3} 0
\]

We now show the independence for the construction.

Lemma 3.2. \(T_i\) and \(T_j\) are ISTs rooted at 0 of \(Q_k\) for \(0 \leq i \neq j \leq k - 1\).

Proof. Let \(x(\neq 0) \in Q_k\) be any vertex with binary string \(x = x_{k-1}x_{k-2} \cdots x_0\). Without loss of generality, we may consider the following three cases:

Case 1. \(i, j \in H_c\). Assume that \(p\) and \(q\) are two distinct rows of \(I_k(x)\) containing \(i\) and \(j\) as their last elements, respectively. Let \(s_1\) and \(s_2\) be any two strict prefix-powersums of row \(p\) and of row \(q\), respectively. By Proposition 2.1, \(s_1 \neq s_2\). Thus, \(x - s_1 \neq x - s_2\) and it further implies that \(T_i[x, 0] || T_j[x, 0]\). (Refer to \(T_2[13, 0]\) and \(T_3[13, 0]\) in Example 3 as the case.)

Case 2. \(i \in H_c\) and \(j \not\in H_c\). Assume that \(p\) is the row of \(I_k(x)\) that contains \(i\) as the last element. Let \(y = x + 2^i\) and suppose that the row \(q\) in \(I_k(y)\) contains \(j\) as the last element. Let \(s_1\) and \(s_2\) be any two strict prefix-powersums of row \(p\) and of row \(q\), respectively. Clearly, \(x - s_1 \neq y\). Moreover, by Proposition 2.2, we have \(x - s_1 \neq y - s_2\). Thus, \(T_i[x, 0] || T_j[x, 0]\). (Refer to \(T_0[13, 0]\) and \(T_1[13, 0]\) in Example 3 as the case.)

Case 3. \(i, j \not\in H_c\). Let \(y = x + 2^i\) and \(z = x + 2^j\). Assume that the row \(p\) in \(I_k(y)\) contains \(i\) as the last element and the row \(q\) in \(I_k(z)\) contains \(j\) as the last element. Let \(s_1\) and \(s_2\) be any two strict prefix-powersums of row \(p\) and of row \(q\), respectively. Clearly, \(y - s_1 \neq z - s_2\). Thus, \(T_i[x, 0] || T_j[x, 0]\). (Refer to \(T_0[6, 0]\) and \(T_3[6, 0]\) in Example 3 as the case.)

Since we arbitrarily choose \(x\) from \(Q_k\), \(T_i\) and \(T_j\) are independent. \(\square\)

Combining Lemmas 3.1 and 3.2 gives the following theorem.

Theorem 1. The algorithm of constructing \(k\) ISTs of \(Q_k\) described above can be done in \(O(kN)\) time, where \(N = 2^k\). In particular, the algorithm can be parallelized on \(Q_k\) to run in \(O(k)\) time.

4. Concluding remarks

In this paper, using the concept of Hamming distance Latin square, we design an algorithm to generate \(k\) ISTs of \(Q_k\). Comparing with the recursive construction of Tang et al. [20], our generation is more simple and is easy to be parallelized. Especially, an advantage of our algorithm is the rather small memory requirement when the computation is on a single processor. The space used in our algorithm is \(O(kN)\), which approximates to half of the amount of memory required in [20] (The total amount of memory used during the recursive construction is \(\sum_{i=1}^{k-1} i \cdot 2^i = (k - 1)2^{i+1} + 2 = (2k - 2)N + 2\)). Furthermore, due to the symmetry of the structure of \(Q_k\) and the unification of the constructing rule, the spanning trees we generated have several emphatic features. There is a natural correspondence between the labels of vertices in any two ISTs. The label of a vertex in the spanning tree \(T_i\) can be obtained from the label in the same position of \(T_{i-1}\) by rotating one bit to the
left, where the indices $i$ and $i - 1$ are taken modulo $k$. Thus, if we ignore the representation of binary strings of vertices, then all the resulting $k$ ISTs are isomorphic. This advantage can unify the design of broadcasting scheme for data communication.

Also, we observe that the bound of the height for each IST of $Q_k$ is at least $k + 1$ since the root $0$ in any IST can only be adjacent to a vertex $2^i$ for some $i = 0, 1, \ldots, k - 1$ and the vertex with binary string setting all bits to $1$ except the $i$th bit has the distance $k$ to the vertex $2^i$ in $Q_k$. As to our construction, by Lemma 3.1 we are easy to compute the height of $T_i$ by checking the unique path from a vertex with the farthest distance to the root. As a consequence, the height of $T_i$ constructed from our algorithm attains to the low bound. Moreover, since the length of the unique path from every vertex $x \notin \{0, 2^i\}$ to the vertex $2^i$ in $T_i$ equals to the Hamming distance between $x$ and $2^i$ in $Q_k$, we conclude that $T_i$ has the shortest path length. It further implies that our construction competes with the result of [20] and is optimal in the sense of average path length.

References