# Constructing Independent Spanning Trees on Locally Twisted Cubes in Parallel*

Yu-Huei Chang[2] Jinn-Shyong Yang[1,†] Jou-Ming Chang[2]

[1] Department of Information Management,
National Taipei College of Business, Taipei, Taiwan, ROC

[2] Institute of Information and Decision Sciences,
National Taipei College of Business, Taipei, Taiwan, ROC

## Abstract

Let $LTQ_n$ denote the $n$-dimensional locally twisted cube. Hsieh and Tu (2009) [13] presented an algorithm to construct $n$ edge-disjoint spanning trees rooted at vertex 0 in $LTQ_n$. Later on, Lin et al. (2010) [23] proved that Hsieh and Tu's spanning trees are indeed independent spanning trees (ISTs for short), i.e., all spanning trees are rooted at the same vertex $r$ and for any other vertex $v(\neq r)$, the paths from $v$ to $r$ in any two trees are vertex-disjoint except the two end vertices $v$ and $r$. Shortly afterwards, Liu et al. (2011) [24] pointed out that $LTQ_n$ fails to be vertex-transitive for $n \geqslant 4$ and proposed an algorithm for constructing $n$ ISTs rooted at an arbitrary vertex of $LTQ_n$. Although this algorithm can simultaneously construct $n$ ISTs in parallel, it is not fully parallelized for the construction of each spanning tree. In this paper, we revisit the problem of constructing $n$ ISTs rooted at an arbitrary vertex of $LTQ_n$. As a consequence, we present a fully parallelized approach that is obtained from Hsieh and Tu's algorithm with a slight modification.

**Keyword:** independent spanning trees; edge-disjoint spanning trees; locally twisted cubes; interconnection networks; fault-tolerant broadcasting;

## 1 Introduction

Interconnection networks are usually modeled as undirected simple graphs $G = (V, E)$, where the vertex set $V(= V(G))$ represents the set of processing elements and the edge set $E(= E(G))$ represents the set of communication channels, respectively. A *tree* is a connected graph without cycle. A *rooted tree* is a tree with a distinguished vertex called the root. A subgraph $T$ in a graph $G$ is called a *spanning tree* if $T$ is a tree such that $V(T) = V(G)$. Let $\mathscr{T}$ be a set of $k$ spanning trees of $G$ rooted at a vertex $r$. We say that $\mathscr{T}$ is *edge-disjoint* if the paths from any vertex $v(\neq r)$ to $r$ on the $k$ trees share no common directed edges. By contrast, $\mathscr{T}$ is said to be *independent* if the paths from any vertex $v(\neq r)$ to $r$ on the $k$ trees have no common vertex except $x$ and $y$ (i.e., the paths are *internally vertex-disjoint*).

Constructing independent spanning trees (ISTs for short) in networks have been studied from not only the theoretical point of view but also some practical applications such as fault-tolerant broadcasting [1, 19] and secure message distribution [1, 31, 40]. Let $G$ be a graph and denote $G - F$ the graph obtained from $G$ by removing a set of vertices $F$. A graph $G$ is $k$-*connected* if $|V(G)| > k$ and $G - F$ is connected for every subset $F \subseteq V(G)$ with $|F| < k$. A conjecture proposed by Zehavi and Itai [49] says that any $k$-connected graph has $k$ ISTs rooted at an arbitrary vertex $r$. Henceforth, we refer the conjecture as the IST-Conjecture. From then on, the IST-Conjecture has been confirmed only for $k$-connected graphs with $k \leqslant 4$ (see [19] for $k = 2$, [8,49] for $k = 3$, and [9] for $k = 4$), and it is still open for $k$-connected graphs with $k \geqslant 5$. In addition, by providing construction schemes of ISTs, the IST-Conjecture has been agreed for several restricted classes of graphs or digraphs. For example, the graph classes related to planarity [16, 17, 27, 28], graph classes defined by Cartesian product [3, 29, 32, 33, 36, 42, 46], special classes of digraphs [10, 12, 18, 37], variations of hypercubes [4–7, 24, 34, 35, 40], subclasses of Cayley graphs [21,22,31,41,44,45], and chordal ring [20,43].

The family of locally twisted cubes was first introduced by Yang et al. [47] as a variation of

*This research was partially supported by National Science Council under the Grants NSC102-2221-E-141-002 and NSC102-2221-E-141-001-MY3.

†Corresponding author : shyong@mail.ntcb.edu.tw

hypercube architecture to achieving the improvement in diameter. Hsieh and Tu [13] studied the construction of edge-disjoint spanning trees on locally twisted cubes. Since the $n$-dimensional locally twisted cube $LTQ_n$ is $n$-connected, they presented an algorithm to construct $n$ edge-disjoint spanning trees rooted at vertex 0. At a later time, Lin et al. [23] proved that Hsieh and Tu's spanning trees are indeed independent. Liu et al. [24] pointed out that $LTQ_n$ fails to be vertex-transitive for $n \geqslant 4$ and it does satisfy the even-odd-vertex-transitive property. Thus, the proof of [23], together with Hsieh and Tu's algorithm [13], does not solve the IST-Conjecture on $LTQ_n$. Furthermore, Liu et al. [24] proposed an algorithm for constructing $n$ ISTs rooted at an arbitrary vertex of $LTQ_n$, and thus confirmed the IST-Conjecture for $LTQ_n$. Although the algorithm in [24] can simultaneously construct $n$ ISTs in parallel, it is not fully parallelized for the construction of each spanning tree (in fact, it looks like a constructing scheme of binomial tree in a recursive fashion). In this paper, with a slight modification from Hsieh and Tu's algorithm, we present a fully parallelized approach for constructing $n$ ISTs rooted at an arbitrary vertex of $LTQ_n$.

The rest of this paper is organized as follows. Section 2 formally gives the definition of locally twisted cubes and introduces our constructing scheme of ISTs for $LTQ_n$. Section 3 shows the correctness by proving the independency of the constructed spanning trees. The final section contains our concluding remarks.

# 2 Constructing ISTs on $LTQ_n$ in parallel

Let $\oplus$ denote the modulo 2 addition. For $n \geqslant 2$, the *$n$-dimensional locally twisted cube* $LTQ_n$ is a graph with $\{0,1\}^n$ as its vertex set, and two vertices $x = x_{n-1}x_{n-2}\cdots x_0$ and $y = y_{n-1}y_{n-2}\cdots y_0$ are adjacent in $LTQ_n$ if and only if either

(1) there is an integer $i \in \{2,3,\ldots,n-1\}$ such that $x_i = \bar{y}_i$ and $x_{i-1} = y_{i-1} \oplus x_0$, and $x_j = y_j$ for all remaining bits, or

(2) there is an integer $i \in \{0,1\}$ such that $x_i = \bar{y}_i$, and $x_j = y_j$ for all remaining bits.

If one of the above conditions is fulfilled, then $y$ is called the *$i$-neighbor* of $x$ and is denoted by $y = N_i(x)$. Figure 1 shows the graphs $LTQ_3$ and $LTQ_4$, respectively. The locally twisted cube can be equivalently defined by the following recursive fashion:

(1) $LTQ_2$ is a graph consisting of four vertices labeled with 00, 01, 10, and 11, respectively, connected by four edges (00,01), (00,10), (01,11), and (10,11).

(2) For $n \geqslant 3$, $LTQ_n$ is constructed from two disjoint copies of $LTQ_{n-1}$ according to the following steps: Denote $0LTQ_{n-1}$ (respectively, $1LTQ_{n-1}$) the graph obtained by prefixing the label of each vertex in one copy of $LTQ_{n-1}$ with 0 (respectively, 1). Each vertex $x = 0x_{n-2}x_{n-3}\cdots x_0$ in $0LTQ_{n-1}$ is connected with the vertex $1(x_{n-2}\oplus x_0)x_{n-3}\cdots x_0$ in $1LTQ_{n-1}$ by an edge.

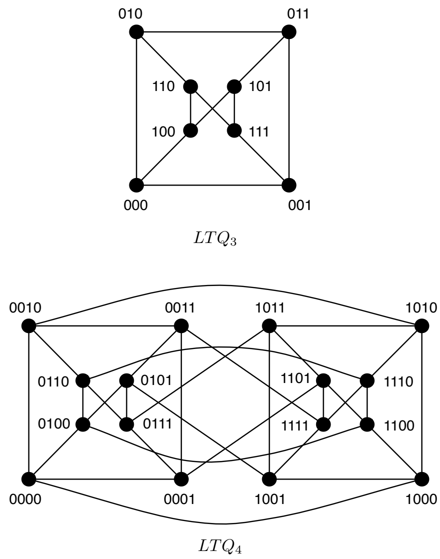


Figure 1: Locally twisted cubes $LTQ_3$ and $LTQ_4$.

From the above definition, it is clear that $LTQ_n$ is an $n$-regular graph, and the binary strings of any two adjacent vertices in $LTQ_n$ differ in at most two successive bits. Yang et al. [47] showed that $LTQ_n$ has a connectivity of $n$. Also, from [24], we know that $LTQ_n$ possesses the property of *even-odd-vertex-transitive*, i.e., for every pair of vertices $x = x_{n-1}x_{n-2}\cdots x_0$ and $y = y_{n-1}y_{n-2}\cdots y_0$ with the same parity (i.e., $x_0 = y_0$), there is an automorphism that maps $x$ to $y$. More previous results on $LTQ_n$ can be found in the literature, e.g., the studies of diagnosability [39], mesh embedding [11], fault-hamiltonicity [14], panconnectivity [25], pancyclicity and fault-pancyclicity [2, 15, 26, 30, 38, 48].

In this paper, we also use the following notation. Two paths $P$ and $Q$ joining two distinct vertices $x$ and $y$ are *internally vertex-disjoint*, denoted by $P||Q$, if $V(P) \cap V(Q) = \{x, y\}$. Let $T$ be a spanning tree rooted at a vertex $r$ of $LTQ_n$. The parent of a vertex $x(\neq r)$ in $T$ is denoted by $\textsc{parent}(T, x)$. For $x, y \in V(T)$, the unique path from $x$ to $y$ is denoted by $T[x, y]$. Hence, two spanning trees $T$ and $T'$ with the same root $r$ are ISTs if and only if $T[x, r] \,||\, T'[x, r]$ for every vertex $x \in V(T) \setminus \{r\}$.

Since $LTQ_n$ has connectivity $n$, the root in each spanning tree must have a unique child. Let $\mathbb{Z}_n = \{0, 1, \ldots, n-1\}$. For $i \in \mathbb{Z}_n$, we denote $T_i$ as the tree such that the root $r$ takes its $i$-neighbor $N_i(r)$ as the unique child. Let $N_i(r) = c_{n-1}c_{n-2}\cdots c_0$. For each vertex $x = x_{n-1}x_{n-2}\cdots x_0 \in V(T_i) \setminus \{r\}$, we define $I_i(x) = \{j \in \mathbb{Z}_n \colon x_j \neq c_j\}$ and $\alpha_i(x) = |\{j \in I_i(x) \colon j > i\}|$. Moreover, if $i \neq 0$, $c_0 = 1$ and $\alpha_i(x)$ is odd, we let $H_i(x) = (I_i(x) \cup \{i\}) \setminus (I_i(x) \cap \{i\})$; otherwise, let $H_i(x) = I_i(x)$. Also, we define the following function:

$$\textsc{next}(i, x) =$$

$$\begin{cases} i & \text{if } H_i(x) = \emptyset; \\ \max H_i(x) & \text{if } H_i(x) \neq \emptyset \text{ and } i < \min H_i(x); \\ \max\{j \in H_i(x) \colon j \leqslant i\} & \text{otherwise.} \end{cases}$$

That is, we regard $H_i(x)$ as a cyclic ordered set in decreasing order. If $H_i(x) = \emptyset$ or $i \in H_i(x)$, the function outputs $i$; otherwise, the function outputs the next element in the cyclic order of $H_i(x)$ with respect to $i$.

In what follows, we present a fully parallelized algorithm for constructing $n$ spanning trees with an arbitrary vertex $r = r_{n-1}r_{n-2}\cdots r_0$ as their common root in $LTQ_n$. For each vertex $x \in V(LTQ_n) \setminus \{r\}$ with binary string $x = x_{n-1}x_{n-2}\cdots x_0$, the construction can be carried out by describing the parent of $x$ in each spanning tree $T_i$.

---

**Algorithm** Constructing-ISTs
**Input:** All vertices of $LTQ_n$ and an arbitrary root $r = r_{n-1}r_{n-2}\cdots r_0$.
**Output:** $n$ ISTs $T_0, T_1, \ldots, T_{n-1}$ root at $r$.
1: **for** $i = 0$ to $n-1$ **do in parallel** /* construct $T_i$ simultaneously */
2: **for** each vertex $x = x_{n-1}x_{n-2}\cdots x_0$ in $LTQ_n$ **do in parallel** /* generate parent of each vertex $x$ simultaneously */
3: $j = \textsc{next}(i, x)$
4: **if** $j \geqslant 2$ and $x_0 = 1$ **then**
5: $\textsc{parent}(T_i, x) = x + (-1)^{x_j} \times 2^j + (-1)^{x_{j-1}} \times 2^{j-1}$
6: **else**
7: $\textsc{parent}(T_i, x) = x + (-1)^{x_j} \times 2^j$

---

Figure 2: Algorithm for constructing $n$ spanning trees in $LTQ_n$.

**Example 1**. Consider $LTQ_4$ and suppose we choose $r = 1011_2 = 11$ as the common root. For conciseness, we represent a vertex in $LTQ_4$ by decimal. We describe how the Constructing-ISTs algorithm constructs $T_2$ in $LTQ_4$ as follows. Clearly, the 2-neighbor of $r$ is $N_2(11) = c_3c_2c_1c_0 = 1101_2 = 13$. Let $x (\neq r)$ be an arbitrary vertex in $LTQ_4$ with $x = x_3x_2x_1x_0$. If $0 \leqslant x \leqslant 7$, then $\alpha_2(x) = 1$; otherwise, $\alpha_2(x) = 0$. Since $i = 2$ and $c_0 = 1$, we have $H_2(x) = I_2(x) \setminus \{2\}$ for $x \in \{0, 1, 2, 3\}$ and $H_2(x) = I_2(x) \cup \{2\}$ for $x \in \{4, 5, 6, 7\}$. Also, we have $H_2(x) = I_2(x)$ for $8 \leqslant x \leqslant 15$ and $x \neq 11$. Consequently, we can determine $j = 0$ when $x \in \{0, 12\}$; $j = 1$ when $x \in \{2, 3, 14, 15\}$; $j = 3$ when $x = 1$; and $j = 2$ otherwise. For $x \in \{1, 5, 7, 9, 13\}$, since $j \geqslant 2$ and $x_0 = 1$, we have $\textsc{parent}(T_2, x) = x + (-1)^{x_j} \times 2^j + (-1)^{x_{j-1}} \times 2^{j-1}$ according to Line 5 of the algorithm. Otherwise, we have $\textsc{parent}(T_2, x) = x + (-1)^{x_j} \times 2^j$ according to Line 7 of the algorithm. Table 1 summarizes the information for constructing $T_i$ for $0 \leqslant i \leqslant 3$ in $LTQ_4$.

Figure 3 illustrates the construction for $LTQ_4$ using Algorithm Constructing-ISTs. For convenience, we adopt the notation $x \xrightarrow{\pm 2^i} y$ (respectively, $x \xrightarrow{\pm 2^i \pm 2^{i-1}} y$) to mean that $x \pm 2^i = y$ (respectively, $x \pm 2^i \pm 2^{i-1} = y$) and $x$ and $y$ are adjacent in $LTQ_n$. For instance, we have $T_2[6, 11] = 6 \xrightarrow{-2^2} 2 \xrightarrow{-2^1} 0 \xrightarrow{+2^0} 1 \xrightarrow{+2^3+2^2} 13 \xrightarrow{-2^2+2^1} 11$ in Figure 3.

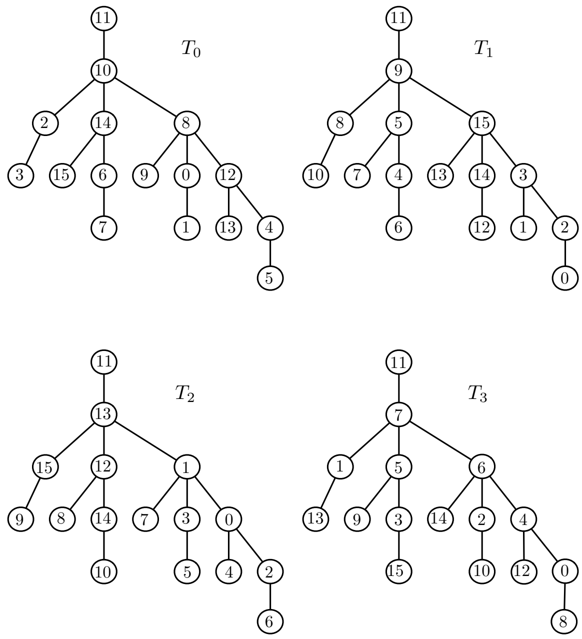


Figure 3: Four ISTs of $LTQ_4$.

# 3 Correctness

In this section, we will show the validity of the algorithm. Firstly, we prove the reachability between every vertex $x (\neq r)$ and the vertex $r$ in $T_i$, thereby proving the existence of a unique path from $x$ to the root in the tree.

**Theorem 1.** *Let $r$ be an arbitrary vertex of $LTQ_n$. The construction of $T_i$ for $i \in \mathbb{Z}_n$ are spanning trees with a common root at $r$.*

Table 1: The parent of vertices $x \in V(LTQ_4) \setminus \{11\}$ in $T_i$ with root $r = 1011_2 = 11$.

$i = 3,\quad N_3(11) = 0111_2 = 7$

| $x$ | binary string | $I_3(x)$ | $\alpha_3(x)$ | $H_3(x)$ | $j = \text{NEXT}(3, x)$ | $x_0$ | $x_j$ | $x_{j-1}$ | $\text{PARENT}(T_3, x)$ |
|---|---|---|---|---|---|---|---|---|---|
| 0 | 0000 | $\{0,1,2\}$ | even | $\{0,1,2\}$ | 2 | 0 | 0 | 0 | $= 0 + 2^2 = 4$ |
| 1 | 0001 | $\{1,2\}$ | even | $\{1,2\}$ | 2 | 1 | 0 | 0 | $= 1 + 2^2 + 2^1 = 7$ |
| 2 | 0010 | $\{0,2\}$ | even | $\{0,2\}$ | 2 | 0 | 0 | 1 | $= 2 + 2^2 = 6$ |
| 3 | 0011 | $\{2\}$ | even | $\{2\}$ | 2 | 1 | 0 | 1 | $= 3 + 2^2 - 2^1 = 5$ |
| 4 | 0100 | $\{0,1\}$ | even | $\{0,1\}$ | 1 | 0 | 0 | 0 | $= 4 + 2^1 = 6$ |
| 5 | 0101 | $\{1\}$ | even | $\{1\}$ | 1 | 1 | 0 | 1 | $= 5 + 2^1 = 7$ |
| 6 | 0110 | $\{0\}$ | even | $\{0\}$ | 0 | 0 | 0 | - | $= 6 + 2^0 = 7$ |
| 7 | 0111 | $\emptyset$ | even | $\emptyset$ | 3 | 1 | 0 | 1 | $= 7 + 2^3 - 2^2 = 11$ |
| 8 | 1000 | $\{0,1,2,3\}$ | even | $\{0,1,2,3\}$ | 3 | 0 | 1 | 0 | $= 8 - 2^3 = 0$ |
| 9 | 1001 | $\{1,2,3\}$ | even | $\{1,2,3\}$ | 3 | 1 | 1 | 0 | $= 9 - 2^3 + 2^2 = 5$ |
| 10 | 1010 | $\{0,2,3\}$ | even | $\{0,2,3\}$ | 3 | 0 | 1 | 0 | $= 10 - 2^3 = 2$ |
| 12 | 1100 | $\{0,1,3\}$ | even | $\{0,1,3\}$ | 3 | 0 | 1 | 1 | $= 12 - 2^3 = 4$ |
| 13 | 1101 | $\{1,3\}$ | even | $\{1,3\}$ | 3 | 1 | 1 | 1 | $= 13 - 2^3 - 2^2 = 1$ |
| 14 | 1110 | $\{0,3\}$ | even | $\{0,3\}$ | 3 | 0 | 1 | 1 | $= 14 - 2^3 = 6$ |
| 15 | 1111 | $\{3\}$ | even | $\{3\}$ | 3 | 1 | 1 | 1 | $= 15 - 2^3 - 2^2 = 3$ |

$i = 2,\quad N_2(11) = 1101_2 = 13$

| $x$ | binary string | $I_2(x)$ | $\alpha_2(x)$ | $H_2(x)$ | $j = \text{NEXT}(2, x)$ | $x_0$ | $x_j$ | $x_{j-1}$ | $\text{PARENT}(T_2, x)$ |
|---|---|---|---|---|---|---|---|---|---|
| 0 | 0000 | $\{0,2,3\}$ | odd | $\{0,3\}$ | 0 | 0 | 0 | - | $= 0 + 2^0 = 1$ |
| 1 | 0001 | $\{2,3\}$ | odd | $\{3\}$ | 3 | 1 | 0 | 0 | $= 1 + 2^3 + 2^2 = 13$ |
| 2 | 0010 | $\{0,1,2,3\}$ | odd | $\{0,1,3\}$ | 1 | 0 | 1 | 0 | $= 2 - 2^1 = 0$ |
| 3 | 0011 | $\{1,2,3\}$ | odd | $\{1,3\}$ | 1 | 1 | 1 | 1 | $= 3 - 2^1 = 1$ |
| 4 | 0100 | $\{0,3\}$ | odd | $\{0,2,3\}$ | 2 | 0 | 1 | 0 | $= 4 - 2^2 = 0$ |
| 5 | 0101 | $\{3\}$ | odd | $\{2,3\}$ | 2 | 1 | 1 | 0 | $= 5 - 2^2 + 2^1 = 3$ |
| 6 | 0110 | $\{0,1,3\}$ | odd | $\{0,1,2,3\}$ | 2 | 0 | 1 | 1 | $= 6 - 2^2 = 2$ |
| 7 | 0111 | $\{1,3\}$ | odd | $\{1,2,3\}$ | 2 | 1 | 1 | 1 | $= 7 - 2^2 - 2^1 = 1$ |
| 8 | 1000 | $\{0,2\}$ | even | $\{0,2\}$ | 2 | 0 | 0 | 0 | $= 8 + 2^2 = 12$ |
| 9 | 1001 | $\{2\}$ | even | $\{2\}$ | 2 | 1 | 0 | 0 | $= 9 + 2^2 + 2^1 = 15$ |
| 10 | 1010 | $\{0,1,2\}$ | even | $\{0,1,2\}$ | 2 | 0 | 0 | 1 | $= 10 + 2^2 = 14$ |
| 12 | 1100 | $\{0\}$ | even | $\{0\}$ | 0 | 0 | 0 | - | $= 12 + 2^0 = 13$ |
| 13 | 1101 | $\emptyset$ | even | $\emptyset$ | 2 | 1 | 1 | 0 | $= 13 - 2^2 + 2^1 = 11$ |
| 14 | 1110 | $\{0,1\}$ | even | $\{0,1\}$ | 1 | 0 | 1 | 0 | $= 14 - 2^1 = 12$ |
| 15 | 1111 | $\{1\}$ | even | $\{1\}$ | 1 | 1 | 1 | 1 | $= 15 - 2^1 = 13$ |

$i = 1,\quad N_1(11) = 1001_2 = 9$

| $x$ | binary string | $I_1(x)$ | $\alpha_1(x)$ | $H_1(x)$ | $j = \text{NEXT}(1, x)$ | $x_0$ | $x_j$ | $x_{j-1}$ | $\text{PARENT}(T_1, x)$ |
|---|---|---|---|---|---|---|---|---|---|
| 0 | 0000 | $\{0,3\}$ | odd | $\{0,1,3\}$ | 1 | 0 | 0 | 0 | $= 0 + 2^1 = 2$ |
| 1 | 0001 | $\{3\}$ | odd | $\{1,3\}$ | 1 | 1 | 0 | 1 | $= 1 + 2^1 = 3$ |
| 2 | 0010 | $\{0,1,3\}$ | odd | $\{0,3\}$ | 0 | 0 | 0 | - | $= 2 + 2^0 = 3$ |
| 3 | 0011 | $\{1,3\}$ | odd | $\{3\}$ | 3 | 1 | 0 | 0 | $= 3 + 2^3 + 2^2 = 15$ |
| 4 | 0100 | $\{0,2,3\}$ | even | $\{0,2,3\}$ | 0 | 0 | 0 | - | $= 4 + 2^0 = 5$ |
| 5 | 0101 | $\{2,3\}$ | even | $\{2,3\}$ | 3 | 1 | 0 | 1 | $= 5 + 2^3 - 2^2 = 9$ |
| 6 | 0110 | $\{0,1,2,3\}$ | even | $\{0,1,2,3\}$ | 1 | 0 | 1 | 0 | $= 6 - 2^1 = 4$ |
| 7 | 0111 | $\{1,2,3\}$ | even | $\{1,2,3\}$ | 1 | 1 | 1 | 1 | $= 7 - 2^1 = 5$ |
| 8 | 1000 | $\{0\}$ | even | $\{0\}$ | 0 | 0 | 0 | - | $= 8 + 2^0 = 9$ |
| 9 | 1001 | $\emptyset$ | even | $\emptyset$ | 1 | 1 | 0 | 1 | $= 9 + 2^1 = 11$ |
| 10 | 1010 | $\{0,1\}$ | even | $\{0,1\}$ | 1 | 0 | 1 | 0 | $= 10 - 2^1 = 8$ |
| 12 | 1100 | $\{0,2\}$ | odd | $\{0,1,2\}$ | 1 | 0 | 0 | 0 | $= 12 + 2^1 = 14$ |
| 13 | 1101 | $\{2\}$ | odd | $\{1,2\}$ | 1 | 1 | 0 | 1 | $= 13 + 2^1 = 15$ |
| 14 | 1110 | $\{0,1,2\}$ | odd | $\{0,2\}$ | 0 | 0 | 0 | - | $= 14 + 2^0 = 15$ |
| 15 | 1111 | $\{1,2\}$ | odd | $\{2\}$ | 2 | 1 | 1 | 1 | $= 15 - 2^2 - 2^1 = 9$ |

$i = 0,\quad N_0(11) = 1010_2 = 10$

| $x$ | binary string | $I_0(x)$ | $\alpha_0(x)$ | $H_0(x)$ | $j = \text{NEXT}(0, x)$ | $x_0$ | $x_j$ | $x_{j-1}$ | $\text{PARENT}(T_0, x)$ |
|---|---|---|---|---|---|---|---|---|---|
| 0 | 0000 | $\{1,3\}$ | even | $\{1,3\}$ | 3 | 0 | 0 | 0 | $= 0 + 2^3 = 8$ |
| 1 | 0001 | $\{0,1,3\}$ | even | $\{0,1,3\}$ | 0 | 1 | 1 | - | $= 1 - 2^0 = 0$ |
| 2 | 0010 | $\{3\}$ | odd | $\{3\}$ | 3 | 0 | 0 | 0 | $= 2 + 2^3 = 10$ |
| 3 | 0011 | $\{0,3\}$ | odd | $\{0,3\}$ | 0 | 1 | 1 | - | $= 3 - 2^0 = 2$ |
| 4 | 0100 | $\{1,2,3\}$ | odd | $\{1,2,3\}$ | 3 | 0 | 0 | 1 | $= 4 + 2^3 = 12$ |
| 5 | 0101 | $\{0,1,2,3\}$ | odd | $\{0,1,2,3\}$ | 0 | 1 | 1 | - | $= 5 - 2^0 = 4$ |
| 6 | 0110 | $\{2,3\}$ | even | $\{2,3\}$ | 3 | 0 | 0 | 1 | $= 6 + 2^3 = 14$ |
| 7 | 0111 | $\{0,2,3\}$ | even | $\{0,2,3\}$ | 0 | 1 | 1 | - | $= 7 - 2^0 = 6$ |
| 8 | 1000 | $\{1\}$ | odd | $\{1\}$ | 1 | 0 | 0 | 0 | $= 8 + 2^1 = 10$ |
| 9 | 1001 | $\{0,1\}$ | odd | $\{0,1\}$ | 0 | 1 | 1 | - | $= 9 - 2^0 = 8$ |
| 10 | 1010 | $\emptyset$ | even | $\emptyset$ | 0 | 0 | 0 | - | $= 10 + 2^0 = 11$ |
| 12 | 1100 | $\{1,2\}$ | even | $\{1,2\}$ | 2 | 0 | 1 | 0 | $= 12 - 2^2 = 8$ |
| 13 | 1101 | $\{0,1,2\}$ | even | $\{0,1,2\}$ | 0 | 1 | 1 | - | $= 13 - 2^0 = 12$ |
| 14 | 1110 | $\{2\}$ | odd | $\{2\}$ | 2 | 0 | 1 | 1 | $= 14 - 2^2 = 10$ |
| 15 | 1111 | $\{0,2\}$ | odd | $\{0,2\}$ | 0 | 1 | 1 | - | $= 15 - 2^0 = 14$ |

**Proof.** From CONSTRUCTING-ISTS, we know that every vertex $v \in V(LTQ_n)$ implies $v \in T_i$. It follows that $T_i$ is a spanning subgraph of $LTQ_n$. Hereafter, for every vertex $v \in V(LTQ_n) \setminus \{r\}$, all indices of elements of $H_i(v)$ are taken modulo $|H_i(v)|$. Let $x = x_{n-1}x_{n-2}\cdots x_0$ be any vertex of $LTQ_n$. We claim that $T_i[x, r]$ is the unique path connecting $x$ and $r$ in $T_i$. Clearly, if $H_i(x) = \emptyset$, then $x = N_i(r)$ and $\text{NEXT}(i, x) = i$. If $i \geqslant 2$ and $x_0 = 1$, let $J = (-1)^{x_i} \times 2^i + (-1)^{x_{i-1}} \times 2^{i-1}$; otherwise, let $J = (-1)^{x_i} \times 2^i$. Thus, $T_i[x, r] = x \xrightarrow{J} r$ is the desired path that connects $x$ and $r$ in $T_i$. Next, we suppose that $H_i(x) = \{j_0, j_1, \ldots, j_{p-1}\}$ is nonempty and $j_0 < j_1 < \cdots < j_{p-1}$. Consider the following two cases:

**Case 1**: $i \notin H_i(x)$. We assume that $j_k = \text{NEXT}(i, x)$ is the next element in the cyclic order of $H_i(x)$ with respect to $i$, where $0 \leqslant k \leqslant p-1$. If $j_k \geqslant 2$ and $x_0 = 1$, let $J_1 = (-1)^{x_{j_k}} \times 2^{j_k} + (-1)^{x_{j_k-1}} \times 2^{j_k-1}$; otherwise, let $J_1 = (-1)^{x_{j_k}} \times 2^{j_k}$. Thus, we have $\text{PARENT}(T_i, x) = x + J_1$. Let $y = y_{n-1}y_{n-2}\cdots y_0 = x + J_1$ be such a vertex and consider the following scenarios for $H_i(y)$. For $j_k \geqslant 2$ and $x_0 = 1$, if $j_{k-1} = j_k - 1$, then $H_i(y) = H_i(x) \setminus \{j_k, j_{k-1}\}$; otherwise, $H_i(y) = (H_i(x) \setminus \{j_k\}) \cup \{j_k - 1\}$. On the other hand (i.e., $j_k \in \{0, 1\}$ or $x_0 = 0$), we have $H_i(y) = H_i(x) \setminus \{j_k\}$. By a similar argument, we let $j_\ell = \text{NEXT}(i, y)$ and $z = z_{n-1}z_{n-2}\cdots z_0 = \text{PARENT}(T_i, y) = y + J_2$, where $J_2 = (-1)^{x_{j_\ell}} \times 2^{j_\ell} + (-1)^{x_{j_\ell-1}} \times 2^{j_\ell-1}$ for $j_\ell \geqslant 2$ and $y_0 = 1$; or $J_2 = (-1)^{x_{j_\ell}} \times 2^{j_\ell}$ otherwise. Again, we can determine $H_i(z)$ according to $j_\ell$, $y_0$ and the condition $j_{\ell-1} = j_\ell - 1$ or $j_{\ell-1} \neq j_\ell - 1$. By this way, we find a sequence of vertices $y, z, \ldots, w$ in $T_i$ such that $H_i(w) = \emptyset$, and thus $w = N_i(r)$. Recall that we have constructed $T_i[w, r] = w \xrightarrow{J} r$ for connecting $N_i(r)$ and $r$ in $T_i$. Therefore, we obtain the following unique path that connects $x$ and $r$ in $T_i$:

$$T_i[x, r] = x \xrightarrow{J_1} y \xrightarrow{J_2} z \xrightarrow{J_3} \cdots \xrightarrow{J_d} w \xrightarrow{J} r.$$

**Case 2**: $i \in H_i(x)$. Suppose $i = j_k$ for some $k \in \{0, 1, \ldots, p-1\}$. In this case, we have $\text{NEXT}(i, x) = i$ by definition. If $i \geqslant 2$ and $x_0 = 1$, let $\hat{J} = (-1)^{x_i} \times 2^i + (-1)^{x_{i-1}} \times 2^{i-1}$; otherwise, let $\hat{J} = (-1)^{x_i} \times 2^i$. Moreover, we let $y = \text{PARENT}(T_i, x) = x + \hat{J}$. Clearly, $j_k \notin H(y)$, and thus $y$ is in the situation of Case 1. Let $P = T_i[y, r]$ be the path connecting $y$ and $r$ in $T_i$. Therefore, we obtain the unique path $T_i[x, r]$ by concatenating $x \xrightarrow{\hat{J}} y$ and $P$. ☐

To show the independency of $T_0, T_1, \ldots, T_{n-1}$, we assume that $p, q \in \mathbb{Z}_n$ are any two distinct integers. Let $x = x_{n-1}x_{n-2}\cdots x_0 \in V(LTQ_n) \setminus \{r\}$ be any vertex and two paths $P = T_p[x, r]$ and $Q = T_q[x, r]$ be constructed in Theorem 1. Without loss of generality, we assume $p > q$. Let $c = N_p(r) = c_{n-1}c_{n-2}\cdots c_0$ and $d = N_q(r) = d_{n-1}d_{n-2}\cdots d_0$. For notational convenience, the subpath of $P$ between two vertices $u, v \in V(P)$ is denoted by $P(u, v)$. Moreover, we write $y_i|P(u, v) = b$ for $i \in \mathbb{Z}_n$ and $b \in \{0, 1\}$ to mean that $y_i$ is assigned to $b$ for every vertex $y = y_{n-1}y_{n-2}\cdots y_0$ in the path $P(u, v)$. Similarly, we can define $Q(u, v)$ and $y_i|Q(u, v) = b$ by the same way. In what follows, we always assume that $y = y_{n-1}y_{n-2}\cdots y_0$ is any vertex in $P \setminus \{x, r\}$ and $z = z_{n-1}z_{n-2}\cdots z_0$ is any vertex in $Q \setminus \{x, r\}$. In particular, let $\hat{y} = \hat{y}_{n-1}\hat{y}_{n-2}\cdots \hat{y}_0$ (respectively, $\hat{z} = \hat{z}_{n-1}\hat{z}_{n-2}\cdots \hat{z}_0$) be the vertex adjacent to $x$ in $P$ (respectively, in $Q$). To show that $P||Q$, it suffices to prove that $P(\hat{y}, c) \cap Q(\hat{z}, d) = \emptyset$. For $P$ and $Q$, we also use underscore to mark the different bits between the two paths.

Note that we omit the proof if one of $P(\hat{y}, c)$ and $Q(\hat{z}, d)$ is a null path (i.e., $H_p(x) = \emptyset$ or $H_q(x) = \emptyset$). Moreover, due to the space limitation, we only prove Lemmas 2, 3, 4 and 5, and omit the following four cases: (i) $c_0 = d_0 = 1$ and $x_0 = 0$; (ii) $c_0 = 1$ and $d_0 = x_0 = 0$; (iii) $c_0 = d_0 = x_0 = 1$; and (iv) $c_0 = x_0 = 1$ and $d_0 = 0$.

**Lemma 2.** *If $c_0 = d_0 = x_0 = 0$, then $P||Q$.*

**Proof.** Since $p > q$ and $c_0 = d_0 = 0$, it implies $r_0 = 0$ and $p > q \neq 0$. Moreover, since $c_0 = d_0 = 0$, it follows that $H_p(x) = I_p(x)$ and $H_q(x) = I_q(x)$. In addition, we have $c_p \neq d_p$, $c_q \neq d_q$ and $c_i = d_i$ for $i \in \mathbb{Z}_n \setminus \{p, q\}$. Let $p' = \text{NEXT}(p, x)$ and $q' = \text{NEXT}(q, x)$. We consider the following three scenarios.

**Case 1**: $x_p \neq c_p$ (i.e., $p \in H_p(x)$). In this case, $p = p'$ and $q \geqslant q'$. Since $x_0 = 0$, by Line 7 of the algorithm, we have $\hat{y} = x + (-1)^{x_p} \times 2^p$. Thus, $\hat{y}_p = c_p$. Moreover, since $T_p$ takes $(-1)^{c_p} \times 2^p$ as the last link to connect the root, it implies $y_p|P(\hat{y}, c) = c_p$. On the other hand, since $x_p \neq c_p$ and $c_p \neq d_p$, we have $x_p = d_p$. Thus, $Q$ never changes the bit $z_p$ in the path and $z_p|Q(\hat{z}, d) = d_p \neq c_p$. As a result, $P||Q$. (For example, consider $P = T_3[0, 4] : 0000 \xrightarrow{+2^3} \underline{1}000 \xrightarrow{+2^2} \underline{1}100 \xrightarrow{-2^3} 0100$ and $Q = T_1[0, 4] : 0000 \xrightarrow{+2^1} \underline{0}010 \xrightarrow{+2^2} \underline{0}110 \xrightarrow{-2^1} 0100$ in $LTQ_4$.)

**Case 2**: $x_q \neq d_q$ (i.e., $q \in H_q(x)$). In this case, $p \geqslant p'$ and $q = q'$. Since $x_q \neq d_q$ and $c_q \neq d_q$, we have $x_q = c_q$. Thus, $P$ never changes the bit $y_q$ in the path and $y_q|P(\hat{y}, c) = c_q$. On the other hand, since $x_0 = 0$, by Line 7 of the algorithm, we have $\hat{z} = x + (-1)^{x_q} \times 2^q$. Thus, $\hat{z}_q = d_q$. Moreover, since $T_q$ takes $(-1)^{d_q} \times 2^q$ as the last link to connect the root, it implies $z_q|Q(\hat{z}, d) = d_q \neq c_q$. As a result, $P||Q$. (For example, consider $P = T_2[12, 4] : 1100 \xrightarrow{-2^2} 10\underline{0}0 \xrightarrow{-2^3} 00\underline{0}0 \xrightarrow{+2^2} 0100$ and $Q = T_1[12, 4] : 1100 \xrightarrow{+2^1} 11\underline{1}0 \xrightarrow{-2^3} 01\underline{1}0 \xrightarrow{-2^1} 0100$ in $LTQ_4$.)

**Case 3**: $x_p = c_p$ and $x_q = d_q$ (i.e., $p \notin H_p(x)$ and $q \notin H_q(x)$). In this case, $p > p'$ and $q > q'$. Clearly, $x_{p'} \neq c_{p'}$ and $x_{q'} \neq d_{q'}$. There are three subcases as follows.

**Case 3.1**: $p' = q$. Since $x_0 = 0$, by Line 7 of the algorithm, we have $\hat{y} = x + (-1)^{x_{p'}} \times 2^{p'}$. Thus, $\hat{y}_{p'} = c_{p'}$. Moreover, since it remains unchanged the bit $y_{p'}$ for every vertex $y \in P(\hat{y}, c)$, we have $y_{p'}|P(\hat{y}, c) = c_{p'}$ (i.e., $y_q|P(\hat{y}, c) = c_q$). On the other hand, since $x_q = d_q$ and $T_q$ takes $(-1)^{d_q} \times 2^q$ as the last link to connect the root, it implies $z_q|Q(\hat{z}, d) = d_q \neq c_q$. Thus, $P||Q$. (For example, consider $P = T_3[10, 4] : 1010 \xrightarrow{+2^2} 1\underline{1}10 \xrightarrow{-2^1} 1\underline{1}00 \xrightarrow{-2^3} 0100$ and $Q = T_2[10, 4] : 1010 \xrightarrow{-2^1} 1\underline{0}00 \xrightarrow{-2^3} 0\underline{0}00 \xrightarrow{+2^2} 0100$ in $LTQ_4$.)

**Case 3.2**: $q' = p$. Since $x_p = c_p$ and $T_p$ takes $(-1)^{c_p} \times 2^p$ as the last link to connect the root, it implies $y_p|P(\hat{y}, c) = c_p$. On the other hand, since $x_0 = 0$, by Line 7 of the algorithm, we have $\hat{z} = x+(-1)^{x_{q'}} \times 2^{q'}$. Thus, $\hat{z}_{q'} = d_{q'}$. Moreover, since it remains unchanged the bit $z_{q'}$ for every vertex $z \in Q(\hat{z}, d)$, we have $z_{q'}|Q(\hat{z}, d) = d_{q'}$ (i.e., $z_p|Q(\hat{z}, d) = d_p \neq c_p$). Thus, $P||Q$. (For example, consider $P = T_3[10, 4] : 1010 \xrightarrow{+2^2} \underline{1}110 \xrightarrow{-2^1} \underline{1}100 \xrightarrow{-2^3} 0100$ and $Q = T_1[10, 4] : 1010 \xrightarrow{-2^3} \underline{0}010 \xrightarrow{+2^2} \underline{0}110 \xrightarrow{-2^1} 0100$ in $LTQ_4$.)

**Case 3.3**: $p' \neq q$ and $q' \neq p$. Clearly, $c_{p'} = d_{p'}$ and $c_{q'} = d_{q'}$. In this case, an argument similar to Case 3.1 shows that $y_{p'}|P(\hat{y}, c) = c_{p'}$. Also, an argument similar to Case 3.2 shows that $y_p|Q(\hat{y}, c) = c_p$. On the other hand, since $x_p = c_p \neq d_p$, $x_{p'} \neq c_{p'} = d_{p'}$ and $d_0 = 0$, we have $p, p' \in H_q(x)$. Let $w(= w_{n-1}w_{n-2} \cdots w_0)$ and $w'$ be vertices on $Q$ such that $w' = w + (-1)^{w_p} \times 2^p$. Since $Q(\hat{z}, w)$ has not dealt with the bit $z_{p'}$ for every vertex $z \in Q(\hat{z}, w)$, we have $z_{p'}|Q(\hat{z}, w) = x_{p'} \neq c_{p'}$. Also, since $Q(w', d)$ has dealt with the bit $z_p$, we have $z_p|Q(w', d) = d_p \neq c_p$. Thus, $P(\hat{y}, c) \cap (Q(\hat{z}, w) \cup Q(w', d)) = \emptyset$ and $P||Q$. (For example, consider $P = T_4[26, 4] : 11010 \xrightarrow{-2^3} \underline{1}0010 \xrightarrow{+2^2} \underline{1}0110 \xrightarrow{-2^1} \underline{1}0100 \xrightarrow{-2^4} 00100$ and $Q = T_2[26, 4] : 11010 \xrightarrow{-2^1} \underline{1}1000 \xrightarrow{-2^4} \underline{01}000 \xrightarrow{-2^3} \underline{00}000 \xrightarrow{+2^2} 00100$ in $LTQ_5$.) $\square$

**Lemma 3.** *If $c_0 = x_0 = 0$ and $d_0 = 1$, then $P||Q$.*

**Proof.** Since $p > q$, $c_0 = x_0 = 0$ and $d_0 = 1$, it implies $r_0 = 0$ and $p > q = q' = 0$, where $q' = \text{NEXT}(q, x)$. Moreover, since $x_0 \neq d_0$, it follows that $0 \in I_0(x)(= H_0(x))$. Since $x_0 = c_0$, $P$ never changes the bit $y_0$ in the path, and thus $y_0|P(\hat{y}, c) = c_0 = 0$. On the other hand, by Line 7 of the algorithm, we have $\hat{z} = x + (-1)^{x_0} \times 2^0 = x + 1$. Thus, $\hat{z}_0 = 1$. Moreover, since it remains unchanged the bit $z_0$ for every vertex $z \in Q(\hat{z}, d)$, we have $z_0|Q(\hat{z}, d) = 1$. Thus, $P||Q$. (For example, consider $P = T_3[10, 4] : 1010 \xrightarrow{+2^2} 111\underline{0} \xrightarrow{-2^1} 110\underline{0} \xrightarrow{-2^3} 0100$ and $Q = T_0[10, 4] : 1010 \xrightarrow{+2^0} 101\underline{1} \xrightarrow{-2^3+2^2} 011\underline{1} \xrightarrow{-2^1} 010\underline{1} \xrightarrow{-2^0} 0100$ in $LTQ_4$.) $\square$

**Lemma 4.** *If $c_0 = d_0 = 0$ and $x_0 = 1$, then $P||Q$.*

**Proof.** Since $p > q$ and $c_0 = d_0 = 0$, it implies $r_0 = 0$ and $p > q \neq 0$. Moreover, since $c_0 = d_0 = 0$, it follows that $H_p(x) = I_p(x)$ and $H_q(x) = I_q(x)$. In addition, we have $c_p \neq d_p$, $c_q \neq d_q$ and $c_i = d_i$ for $i \in \mathbb{Z}_n \setminus \{p, q\}$. Let $p' = \text{NEXT}(p, x)$ and $q' = \text{NEXT}(q, x)$. We consider the following three scenarios.

**Case 1**: $x_p \neq c_p$ (i.e., $p \in H_p(x)$). In this case, $p = p' \geqslant 2$. Since $x_0 = 1$, by Line 5 of the algorithm, we have $\hat{y} = x+(-1)^{x_p} \times 2^p+(-1)^{x_{p-1}} \times 2^{p-1}$. Thus, $\hat{y}_p = c_p$. Moreover, since $T_p$ takes $(-1)^{c_p} \times 2^p$ as the last link to connect the root, it implies $y_p|P(\hat{y}, c) = c_p$. On the other hand, since $x_p \neq c_p$ and $c_p \neq d_p$, we have $x_p = d_p$. Thus, $Q$ never changes the bit $z_p$ in the path and $z_p|Q(\hat{z}, d) = d_p \neq c_p$. As a result, $P||Q$. (For example, consider $P = T_3[1, 4] : 0001 \xrightarrow{+2^3+2^2} \underline{1}101 \xrightarrow{-2^0} \underline{1}100 \xrightarrow{-2^3} 0100$ and $Q = T_1[1, 4] : 0001 \xrightarrow{+2^1} \underline{0}011 \xrightarrow{-2^0} \underline{0}010 \xrightarrow{+2^2} \underline{0}110 \xrightarrow{-2^1} 0100$ in $LTQ_4$.)

**Case 2**: $x_p = c_p$ and $x_q \neq d_q$ (i.e., $p \notin H_p(x)$ and $q \in H_q(x)$). In this case, $p > p'$ and $q = q'$. Moreover, we have $x_{p'} \neq c_{p'}$. Since $x_q \neq d_q$ and $c_q \neq d_q$, it implies $x_q = c_q$, and thus $p' \neq q$ and $c_{p'} = d_{p'}$. There are two subcases as follows.

**Case 2.1**: $p' > q$. In this case, we have $p' \geqslant 2$. Since $x_0 = 1$, by Line 5 of the algorithm, we have $\hat{y} = x + (-1)^{x_{p'}} \times 2^{p'} + (-1)^{x_{p'-1}} \times 2^{p'-1}$. Thus, $\hat{y}_{p'} = c_{p'}$. Moreover, since $P$ never changes the bit $y_{p'}$ in the succedent path again, it follows that $y_{p'}|P(\hat{y}, c) = c_{p'}$. Also, since $T_p$ takes $(-1)^{c_p} \times 2^p$ as the last link to connect the root, it implies $y_p|P(\hat{y}, c) = c_p$. On the other hand, let $w(= w_{n-1}w_{n-2} \cdots w_0)$ and $w'$ be vertices on $Q$ such that $w' = w + (-1)^{w_p} \times 2^p$. Since $Q(\hat{z}, w)$ has not dealt with the bit $z_{p'}$ for every vertex $z \in Q(\hat{z}, w)$, we have $z_{p'}|Q(\hat{z}, w) = x_{p'} \neq c_{p'}$. Also, since $Q(w', d)$ has dealt with the bit $z_p$, we have $z_p|Q(w', d) = d_p \neq c_p$. Thus, $P(\hat{y}, c) \cap (Q(\hat{z}, w) \cup Q(w', d)) = \emptyset$ and $P||Q$. (For example, consider $P = T_3[9, 4] : 1001 \xrightarrow{+2^2+2^1} \underline{1}1\underline{1}1 \xrightarrow{-2^1} \underline{1}1\underline{0}1 \xrightarrow{-2^0} \underline{1}1\underline{0}0 \xrightarrow{-2^3} 0100$ and $Q = T_1[9, 4] : 1001 \xrightarrow{+2^1} 1\underline{0}11 \xrightarrow{-2^0} 1\underline{0}10 \xrightarrow{-2^3} \underline{0}010 \xrightarrow{+2^2} \underline{0}110 \xrightarrow{-2^1} 0100$ in $LTQ_4$.)

**Case 2.2**: $q > p'$. Since $x_q = c_q$ and $p > q = q' > p'$, the path $P$ never changes the bit $y_q$. Thus, $y_q|P(\hat{y}, c) = x_q$. On the other hand, since $x_0 = 1$, by Line 5 of the algorithm, we have $\hat{z} = x + (-1)^{x_q} \times 2^q + (-1)^{x_{q-1}} \times 2^{q-1}$. Thus, $\hat{z}_q = d_q$. Moreover, $T_q$ takes $(-1)^{d_q} \times 2^q$ as the last link to connect the root, it follows that $z_q|Q(\hat{z}, d) = d_q \neq x_q$. As a result, $P||Q$. (For example, consider $P = T_3[15, 4] : 1111 \xrightarrow{-2^1} 1\underline{1}01 \xrightarrow{-2^0} 1\underline{1}00 \xrightarrow{-2^3} 0100$ and $Q = T_2[15, 4] : 1111 \xrightarrow{-2^3-2^2} 1\underline{0}01 \xrightarrow{-2^0} 1\underline{0}00 \xrightarrow{-2^3} 0\underline{0}00 \xrightarrow{+2^2} 0100$ in $LTQ_4$.)

**Case 3**: $x_p = c_p$ and $x_q = d_q$ (i.e., $p \notin H_p(x)$ and $q \notin H_q(x)$). Since $x_0 \neq d_0$, it follows that $p > q > q' \geqslant 0$, and thus $q' \neq p$. Moreover, we have $x_{p'} \neq c_{p'}$ and $x_{q'} \neq d_{q'}$. There are two subcases as follows.

**Case 3.1**: $p' = q$. By Line 5 or Line 7 of the algorithm, $P$ first changes the bit $x_{p'}$. Thus, $\hat{y}_{p'} = c_{p'}$. Moreover, since $P$ never changes the bit $y_{p'}$ in the succedent path again, it follows that $y_{p'}|P(\hat{y}, c) = c_{p'} = c_q$. On the other hand, since $x_q = d_q$ and $T_q$ takes $(-1)^{d_q} \times 2^q$ as the last link to connect the root, it follows that $z_q|Q(\hat{z}, d) = d_q \neq c_q$. As a result, $P||Q$. (For example, consider $P = T_3[9, 4] : 1001 \xrightarrow{+2^2+2^1} 1\underline{1}11 \xrightarrow{-2^1} 1\underline{1}01 \xrightarrow{-2^0} 1\underline{1}00 \xrightarrow{-2^3} 0100$ and $Q = T_2[9, 4] : 1001 \xrightarrow{-2^0} 1\underline{0}00 \xrightarrow{-2^3} 0\underline{0}00 \xrightarrow{+2^2} 0100$ in $LTQ_4$.)

**Case 3.2**: $p' \neq q$. In this case, the proof is similar to Case 2.1. That is, we can show that $y_{p'}|P(\hat{y}, c) = c_{p'}$ and $y_p|P(\hat{y}, c) = c_p$. On the other hand, $z_{p'}|Q(\hat{z}, w) = x_{p'} \neq c_{p'}$ and $z_p|Q(w', d) = d_p \neq c_p$, where $w(= w_{n-1}w_{n-2} \cdots w_0)$ and $w' = w + (-1)^{w_p} \times 2^p$ are two adjacent vertices on $Q$. (For example, consider $P = T_3[11, 4] : 1011 \xrightarrow{+2^2-2^1} \underline{11}01 \xrightarrow{-2^0} \underline{11}00 \xrightarrow{-2^3} 0100$ and $Q = T_1[11, 4] : 1011 \xrightarrow{-2^0} 1\underline{0}10 \xrightarrow{-2^3} \underline{0}010 \xrightarrow{+2^2} \underline{0}110 \xrightarrow{-2^1} 0100$ in $LTQ_4$.) $\square$

**Lemma 5.** *If $c_0 = 0$ and $d_0 = x_0 = 1$, then $P||Q$.*

**Proof.** Since $p > q$ and $c_0 = 0$, we have $r_0 = 0$. Also, since $r_0 \neq d_0$ and $d_0 = x_0$, it follows that $q = 0$ and $q' \neq q$. Moreover, $c_0 = 0$ and $q = 0$ imply $H_p(x) = I_p(x)$ and $H_0(x) = I_0(x)$. In addition, we have $c_p \neq d_p$, $c_0 \neq d_0$ and $c_i = d_i$ for $i \in \mathbb{Z}_n \setminus \{p, 0\}$. Let $p' = \textsc{next}(p, x)$ and $q' = \textsc{next}(q, x)$. We consider the following two scenarios.

**Case 1**: $x_p \neq c_p$ (i.e., $p \in H_p(x)$). In this case, $p = p'$. Since $c_p \neq d_p$ and $x_p \neq c_p$, it implies $x_p = d_p$. Since $x_{q'} \neq d_{q'}$ , we have $q' \neq p$. There are two subcases as follows.

**Case 1.1**: $p > q'$. In this case, we have $p = p' > q' > q = 0$. Since $p \geqslant 2$ and $x_0 = 1$, by Line 5 of the algorithm, we have $\hat{y} = x + (-1)^{x_p} \times 2^p + (-1)^{x_{p-1}} \times 2^{p-1}$. Thus, $\hat{y}_p = c_p$. Moreover, since $T_p$ takes $(-1)^{c_p} \times 2^p$ as the last link to connect the root, it follows that $y_p|P(\hat{y}, c) = c_p$. On the other hand, since $x_p = d_p$ and $Q$ never changes the bit $z_p$ in the path, we have $z_p|Q(\hat{z}, d) = d_p \neq c_p$. This shows that $P||Q$. (For example, consider $P = T_3[1, 4] : 0001 \xrightarrow{+2^3+2^2} \underline{1}101 \xrightarrow{-2^0} \underline{1}100 \xrightarrow{-2^3} 0100$ and $Q = T_0[1, 4] : 0001 \xrightarrow{+2^2+2^1} \underline{0}111 \xrightarrow{-2^1} \underline{0}101 \xrightarrow{-2^0} 0100$ in $LTQ_4$.)

**Case 1.2**: $q' > p$. In this case, we have $q' > p = p' > q = 0$. Clearly, $P$ first changes the bit $x_p$ in $x$ and takes $(-1)^{c_p} \times 2^p$ as the last link to connect the root of $T_p$. Let $w(= w_{n-1}w_{n-2} \cdots w_0)$ and $w'$ be vertices on $P$ such that $w' = w + (-1)^{w_q} \times 2^q = w + (-1)^{w_0}$. Since $P(\hat{y}, w)$ has not dealt with the bit $y_{q'}$ for every vertex $y \in P(\hat{y}, w)$, we have $y_{q'}|P(\hat{y}, w) = x_{q'} \neq d_{q'}$. Also, since $P(w', c)$ has dealt with the bit $y_q(= y_0)$, we have $y_0|P(w', c) = c_0 = 0$. On the other hand, since $q' \geqslant 2$ and $x_0 = 1$, by Line 5 of the algorithm, we have $\hat{z} = x + (-1)^{x_{q'}} \times 2^{q'} + (-1)^{x_{q'-1}} \times 2^{q'-1}$. Thus, $\hat{z}_{q'} = d_{q'}$. Moreover, since $Q$ never changes the bit $z_{q'}$ in the succedent path again, it follows that $z_{q'}|P(\hat{z}, d) = d_{q'}$. Also, since $x_0 = d_0 = 1$ and $Q$ takes $(-1)^{d_q} \times 2^q(= -1)$ as the last link to connect the root of $T_0$, it follows that $z_0|Q(\hat{z}, d) = d_0 = 1$. Thus, $(P(\hat{y}, w) \cup P(w', c)) \cap Q(\hat{z}, d) = \emptyset$ and $P||Q$. (For example, consider $P = T_2[15, 4] : 1111 \xrightarrow{-2^2-2^1} \underline{1}001 \xrightarrow{-2^0} 100\underline{0} \xrightarrow{-2^3} 000\underline{0} \xrightarrow{+2^2} 0100$ and $Q = T_0[15, 4] : 1111 \xrightarrow{-2^3-2^2} \underline{0}01\underline{1} \xrightarrow{+2^2-2^1} \underline{0}10\underline{1} \xrightarrow{-2^0} 0100$ in $LTQ_4$.)

**Case 2**: $x_p = c_p$ (i.e., $p \notin H_p(x)$). In this case, $p \neq p'$. Clearly, $x_{p'} \neq c_{p'}$. There are three subcases as follows.

**Case 2.1**: $p' = q$. In this case, we have $p > p' = q = 0$. By Line 7 of the algorithm, we have $\hat{y} = x + (-1)^{x_{p'}} \times 2^{p'} = x - 1$. Thus, $\hat{y}_0 = 0$. Moreover, since $P$ never changes the bit $y_0$ in the succedent path again, it follows that $y_0|P(\hat{y}, c) = 0$. On the other hand, since $x_0 = 1$ and $T_q$ takes $(-1)^{d_q} \times 2^q(= -1)$ as the last link to connect the root, it follows that $z_0|Q(\hat{z}, d) = 1$. This shows that $P||Q$. (For example, consider $P = T_1[15, 4] : 1111 \xrightarrow{-2^0} 111\underline{0} \xrightarrow{-2^3} 011\underline{0} \xrightarrow{-2^1} 0100$ and $Q = T_0[15, 4] : 1111 \xrightarrow{-2^3-2^2} 001\underline{1} \xrightarrow{+2^2-2^1} 010\underline{1} \xrightarrow{-2^0} 0100$ in $LTQ_4$.)

**Case 2.2**: $q' = p$. In this case, we have $q' = p > q = 0$. Since $x_p = c_p$ and $T_p$ takes $(-1)^{c_p} \times 2^p$ as the last link to connect the root, it follows that $y_p|P(\hat{y}, c) = c_p$. On the other hand, since $Q$ first changes the bit $x_{q'}(= x_p)$ in $x$ and never changes the bit $z_p$ in the succedent path again, it follows that $z_p|Q(\hat{z}, d) = d_p \neq c_p$. This shows that $P||Q$. (For example, consider $P = T_3[11, 4] : 1011 \xrightarrow{+2^2-2^1} \underline{1}101 \xrightarrow{-2^0} \underline{1}100 \xrightarrow{-2^3} 0100$ and $Q = T_0[11, 4] : 1011 \xrightarrow{-2^3+2^2} \underline{0}111 \xrightarrow{-2^1} \underline{0}101 \xrightarrow{-2^0} 0100$ in $LTQ_4$.)

**Case 2.3**: $p' \neq q$ and $q' \neq p$. In this case, we have $q' > p > p' > q = 0$. Since $P$ first changes the bit $x_{p'}$ in $x$, we have $\hat{y}_{p'} = c_{p'}$. Moreover, since $P$ never changes the bit $y_{p'}$ in the succedent path again, it follows that $y_{p'}|P(\hat{y}, c) = c_{p'}$. Also, since $x_p = c_p$ and $P$ takes $(-1)^{c_p} \times 2^p$ as the last link to connect the root of $T_p$, we have $y_p|P(\hat{y}, c) = c_p$. On the other hand, let $w(= w_{n-1}w_{n-2} \cdots w_0)$ and $w'$ be vertices on $Q$ such that $w' = w + (-1)^{w_p} \times 2^p$. Since $Q(\hat{z}, w)$ has not dealt with the bit $z_{p'}$ for every vertex $z \in Q(\hat{z}, w)$, we have $z_{p'}|Q(\hat{z}, w) = x_{p'} \neq c_{p'}$. Also, since $Q(w', d)$ has dealt with the bit $z_p$, we have $z_p|Q(w', d) = d_p \neq c_p$. This shows that $P||Q$. (For example, consider $P = T_2[11, 4]$ : $1011 \xrightarrow{-2^1} 1\underline{0}\underline{0}1 \xrightarrow{-2^0} 1\underline{0}\underline{0}0 \xrightarrow{-2^3} 0\underline{0}\underline{0}0 \xrightarrow{+2^2} 0100$ and $Q = T_0[11, 4]$ : $1011 \xrightarrow{-2^3+2^2} 01\underline{1}1 \xrightarrow{-2^1} 0\underline{1}01 \xrightarrow{-2^0} 0100$ in $LTQ_4$.) $\square$

From the above lemmas, we conclude that ISTs constructed in this paper are independent. According to Theorem 1 and the result of independency, we obtain the following main theorem.

**Theorem 6.** *Let $N = 2^n$. For $LTQ_n$ and an arbitrary vertex $r \in V(LTQ_n)$, Algorithm* CONSTRUCTING-ISTS *can correctly construct $n$ ISTs rooted at $r$ in $\mathcal{O}(N \log N)$ time. In particular, the algorithm can be parallelized on $LTQ_n$ by using $N$ processors to run in $\mathcal{O}(\log N)$ time.*

# 4 Concluding remarks

This paper provides a parallel construction of ISTs rooted at an arbitrary vertex of locally twisted cubes. Indeed, all ISTs constructed in here are isomorphic to those in [13, 24], which have height $n + 1$. As we have mentioned earlier, many algorithms have been proposed for constructing ISTs on some interconnection networks. However, some of these networks are not vertex-transitive, and thus the desired ISTs are in need of an arbitrary vertex as the root [4–7, 24, 34, 35]. Although most of these algorithms can simultaneously construct ISTs in parallel, the construction of each spanning tree relies on a recursive expansion and is hard to be fully parallelized. To the best of our knowledge, for class of networks without vertex-transitivity, the present paper is the first to propose the fully parallelized approach for constructing ISTs.

# References


[1] F. Bao, Y. Funyu, Y. Hamada, and Y. Igarashi, Reliable broadcasting and secure distributing in channel networks, in: *Proc. of 3rd International Symposium on Parallel Architectures, Algorithms and Networks*, ISPAN'97, Taipei, December 1997, pp. 472–478.

[2] Q.-Y. Chang, M.-J. Ma, and J.-M. Xu, Fault-tolerant pancyclicity of locally twisted cubes (in Chinese), *J. China Univ. Sci. Tech.*, 36 (2006) 607–610.

[3] X.-B. Chen, Parallel construction of optimal independent spanning trees on Cartesian product of complete graphs, *Inform. Process. Lett.*, 111 (2011) 235–238.

[4] B. Cheng, J. Fan, X. Jia, and J. Jia, Parallel construction of independent spanning trees and an application in diagnosis on Möbius cubes, *J. Supercomput.*, 65 (2013) 1279–1301.

[5] B. Cheng, J. Fan, X. Jia, and J. Wang, Dimension-adjacent trees and parallel construction of independent spanning trees on crossed cubes, *J. Parallel Distrib. Comput.*, 73 (2013) 641–652.

[6] B. Cheng, J. Fan, X. Jia, and S. Zhang, Independent spanning trees in crossed cubes, *Inform. Sci.*, 233 (2013) 276–289.

[7] B. Cheng, J. Fan, X. Jia, S. Zhang, and B. Chen, Constructive algorithm of independent spanning trees on Möbius cubes, *Comput. J.*, 56 (2013) 1347–1362.

[8] J. Cheriyan and S.N. Maheshwari, Finding nonseparating induced cycles and independent spanning trees in 3-connected graphs. *J. Algorithms*, 9 (1988) 507–537.

[9] S. Curran, O. Lee, and X. Yu, Finding four independent trees. *SIAM J. Comput.*, 35 (2006) 1023–1058.

[10] Z. Ge and S.L. Hakimi, Disjoint rooted spanning trees with small depths in deBruijn and Kautz graphs, *SIAM J. Comput.*, 26 (1997) 79–92.

[11] Y. Han, J. Fan, S. Zhang, J. Yang, and P. Qjan, Embedding meshes into locally twisted cubes, *Inform. Sci.*, 180 (2010) 3794–3805.

[12] T. Hasunuma and H. Nagamochi, Independent spanning trees with small depths in iterated line graphs, *Discrete Appl. Math.*, 110 (2001) 189–211.

[13] S.-Y. Hsieh and C.-J. Tu, Constructing edge-disjoint spanning trees in locally twisted cubes, *Theoret. Comput. Sci.*, 410 (2009) 926–932

[14] S.-Y. Hsieh and C.-Y. Wu, Edge-fault-tolerant Hamiltonicity of locally twisted cubes under conditional edge faults, *J. Combin. Optim.* 19 (2010) 16–30.

[15] K.S. Hu, S.-S. Yeoh, C. Chen, and L.-H. Hsu, Node-pancyclicity and edge-pancyclicity of hypercube variants, *Inform. Process. Lett.*, 102 (2007) 1–7.

[16] A. Huck, Independent trees in graphs, *Graphs Combin.*, 10 (1994) 29–45.

[17] A. Huck, Independent trees in planar graphs, *Graphs Combin.*, 15 (1999) 29–77.

[18] A. Huck, Independent branching in acyclic digraphs, *Discrete Math*, 199 (1999) 245–249.

[19] A. Itai and M. Rodeh, The multi-tree approach to reliability in distributed networks, *Inform. Comput.*, 79 (1988) 43–59.

[20] Y. Iwasaki, Y. Kajiwara, K. Obokata, and Y. Igarashi, Independent spanning trees of

chordal rings, *Inform. Process. Lett.*, 69 (1999) 155–160.

[21] J.-S. Kim, H.-O. Lee, E. Cheng, and L. Lipták, Optimal independent spanning trees on odd graphs, *J. Supercomputing*, 56 (2011) 212–225.

[22] J.-S. Kim, H.-O. Lee, E. Cheng, and L. Lipták, Independent spanning trees on even networks, *Inform. Sci.*, 181 (2011) 2892–2905.

[23] J.-C. Lin, J.-S. Yang, C.-C. Hsu, and J.-M. Chang, Independent spanning trees vs. edge-disjoint spanning trees in locally twisted cubes, *Inform. Process. Lett.*, 110 (2010) 414–419.

[24] Y-J. Liu, J.K. Lan, W.Y. Chou, and C. Chen, Constructing independent spanning trees for locally twisted cubes, *Theoret. Comput. Sci.*, 412 (2011) 2237–2252.

[25] M.-J. Ma and J.-M. Xu, Panconnectivity of locally twisted cubes, *Appl. Math. Lett.*, 19 (2006) 673–677.

[26] M.-J. Ma and J.-M. Xu, Weak Edge-pancyclicity of locally twisted cubes, *Ars Combin.*, 89 (2008) 89–94.

[27] K. Miura, S. Nakano, T. Nishizeki, and D. Takahashi, A linear-time algorithm to find four independent spanning trees in four connected planar graphs, *Internat. J. Found. Comput. Sci.*, 10 (1999) 195–210.

[28] S. Nagai and S. Nakano, A linear-time algorithm to find independent spanning trees in maximal planar graphs, *IEICE Trans. Fund. Electron. Comm. Comput. Sci.*, E84-A (2001) 1102–1109.

[29] K. Obokata, Y. Iwasaki, F. Bao, and Y. Igarashi, Independent spanning trees of product graphs and their construction, *IEICE Trans. Fund. Electron. Comm. Comput. Sci.*, E79-A (1996) 1894–1903.

[30] J.-H. Park, H.-S. Lim, and H.-C. Kim, Panconnectivity and pancyclicity of hypercube-like interconnection networks with faulty elements, *Theoret. Comput. Sci.*, 377 (2007) 170–180.

[31] A.A. Rescigno, Vertex-disjoint spanning trees of the star network with applications to fault-tolerance and security, *Inform. Sci.*, 137 (2001)259–276.

[32] S.-M. Tang, Y.-L. Wang, and Y.-H. Leu, Optimal independent spanning trees on hypercubes, *J. Inform. Sci. Eng.*, 20 (2004) 143–155.

[33] S.-M. Tang, J.-S. Yang, Y.-L. Wang, and J.-M. Chang, Independent spanning trees on multidimensional torus networks, *IEEE Trans. Comput.*, 59 (2010) 93–102.

[34] Y. Wang, J. Fan, G. Zhou, and X. Jia, Independent spanning trees on twisted cubes, *J. Parallel Distrib. Comput.*, 72 (2012) 58–69.

[35] Y. Wang, J. Fan, X. Jia, and H. Huang, An algorithm to construct independent spanning trees on parity cubes, *Theoret. Comput. Sci.*, 465 (2012) 61–72.

[36] J. Werapun, S. Intakosum, and V. Boonjing, An efficient parallel construction of optimal independent spanning trees on hypercubes, *J. Parallel Distrib. Comput.*, 72 (2012) 1713–1724.

[37] R.W. Whitty, Vertex-disjoint paths and edge-disjoint branchings in directed graphs, *J. Graph Theory*, 11 (1987) 349–358.

[38] X. Xu, W. Zhai, J.-M. Xu, A. Deng, and Y. Yang, Fault-tolerant edge-pancyclicity of locally twisted cubes *Inform. Sci.*, 181 (2011) 2268–2277.

[39] H. Yang and X. Yang, A fast diagnosis algorithm for locally twisted cube multiprocessor systems under the MM* model, *Comput. Math. Appl.*, 53 (2007) 918–926.

[40] J.-S. Yang, H.-C. Chan, and J.-M. Chang, Broadcasting secure messages via optimal independent spanning trees in folded hypercubes, *Discrete Appl. Math.*, 159 (2011) 1254–1263.

[41] J.-S. Yang and J.-M. Chang, Independent spanning trees on folded hyper-stars, *Networks*, 56 (2010) 272–281.

[42] J.-S. Yang and J.-M. Chang, Optimal independent spanning trees on Cartesian product of hybrid graphs, *Comput. J.*, 57 (2014) 93–99.

[43] J.-S. Yang, J.-M. Chang, S.-M. Tang, and Y.-L. Wang, Reducing the height of independent spanning trees in chordal rings, *IEEE Trans. Parallel Distrib. Syst.*, 18 (2007) 644–657.

[44] J.-S. Yang, J.-M. Chang, S.-M. Tang, and Y.-L. Wang, On the independent spanning trees of recursive circulant graphs $G(cd^m, d)$ with $d > 2$, *Theoret. Comput. Sci.*, 410 (2009) 2001–2010.

[45] J.-S. Yang, J.-M. Chang, S.-M. Tang, and Y.-L. Wang, Constructing multiple independent spanning trees on recursive circulant graphs $G(2^m, 2)$, *Int. J. Found. Comput. Sci.*, 21 (2010) 73–90.

[46] J.-S. Yang, S.-M. Tang, J.-M. Chang, and Y.-L. Wang, Parallel construction of optimal independent spanning trees on hypercubes, *Parallel Comput.*, 33 (2007) 73–79.

[47] X. Yang, D.J. Evans, and G.M. Megson, The locally twisted cubes, *Int. J. Comput. Math.*, 82 (2005) 401–413.

[48] X. Yang, G.M. Megson, and D.J. Evans, Locally twisted cubes are 4-pancyclic, *Appl. Math. Lett.*, 17 (2004) 919–925.

[49] A. Zehavi and A. Itai, Three tree-paths, *J. Graph Theory*, 13 (1989) 175–188.