A One-to-Many Parallel Routing Algorithm on a Generalized Recursive Circulant Graph*

Shyue-Ming Tang[1], Jinn–Shyong Yang[2], Jou–Ming Chang[3] and Yue–Li Wang[4]

[1]Department of Psychology and Social Work,
National Defense University, Taipei, Taiwan.
[2]Department of Information Management,
National Taipei College of Business, Taipei, Taiwan.
[3]Institute of Information and Decision Sciences,
National Taipei College of Business, Taipei, Taiwan.
[4]Department of Information Management,
National Taiwan University of Science and Technology, Taipei, Taiwan.

Abstract

Generalized recursive circulant graphs were first proposed in 2012 (See IEEE Transactions on Parallel and Distributed Systems *23, pp.87-93). This graph class is a superclass of recursive circulant graphs.*

The one-to-many routing can achieve the fault-tolerant broadcasting and secure message distribution in a network. It is a common idea that constructing multiple independent spanning trees rooted at one node guarantees the one-to-many routing. In this paper, we propose an algorithm to construct δ independent spanning trees on a generalized recursive circulant graph with degree δ. Moreover, the algorithm makes the one-to-many routing parallelized.

Keyword: *generalized recursive circulant graph, one-to-many parallel routing, internally disjoint paths, independent spanning trees.*

1. Introduction

In [17], Tang et al. applied a multidimensional vertex-labeling approach to extended the class of recursive circulant graphs to a more generalized graph class, called *generalized recursive circulant graphs* (GRC graphs for short). A GRC graph, denoted by $\mathrm{GR}(m_h, m_{h-1}, \ldots, m_1)$, has $n = \Pi_{i=1}^{h} m_i = m_1 \cdot m_2 \cdot \ldots \cdot m_h$ vertices where $m_i \geqslant 2$ for $i = 1, 2, \ldots, h$. Index i is a *dimension* of the labeling system while m_i is the *base* (or called *radix*) of dimension i. Each vertex in the graph can be expressed as an h-tuple $(x_h, x_{h-1}, \ldots, x_1)$ with $0 \leqslant x_i \leqslant m_i - 1$ for $i = 1, 2, \ldots, h$. The h-dimensional representation of a vertex also forms an h-positional numerical representation based on a *mixed radix number system*, i.e., $x_{h(m_h)} \ldots x_{2(m_2)} x_{1(m_1)}$ [9]. For the sake of conciseness, we omit the radices in the numerical representation. In a GRC graph, vertex $(x_h, \ldots, x_{i+1}, x_i, \ x_{i-1}, \ldots, x_1)$ is adjacent to all of those vertices labeled by $(y_h, \ldots, y_{i+1}, y_i, y_{i-1}, \ldots, y_1)$ for $i = 1, 2, \ldots, h$ if and only if $x_j = y_j$ for $j = 1, 2, \ldots, i-1$ and number $y_h \cdots y_{i+1} y_i$ is equal to $x_h \cdots x_{i+1} x_i \pm 1$ in the mixed radix number system. Note that the addition (or subtraction) in each position may cause a carry (or borrow), while the carry (or borrow) occurred at position h will be neglected. This means that, in case of $m_h = 2$, either $x_h + 1$ or $x_h - 1$ obtains the same value and thus contributes to only one adjacent vertex. For example, both $\mathrm{GR}(2, 4, 3)$ and $\mathrm{GR}(3, 2, 4)$ have 24 vertices, as shown in Figure 1. In Figure 1(a), the 5 neighbors of vertex (1,3,0) in $\mathrm{GR}(2, 4, 3)$ are $(1, 3, 0 + 1)$, $(1, 3, 0 - 1)$, $(1, 3 + 1, 0)$, $(1, 3 - 1, 0)$, and $(1 \pm 1, 3, 0)$ which are (1,3,1), (1,2,2), (0,0,0), (1,2,0), and (0,3,0), respectively. In Figure 1(b), the 6 neighbors of vertex (0,1,2) in $\mathrm{GR}(3, 2, 4)$ are (1,1,2), (2,1,2), (1,0,2), (0,0,2), (0,1,3) and (0,1,1).

A GRC graph can be decomposed recursively.

*This research is supported by the National Science Council of Taiwan under the Grants NSC102–2410–H–606–005.

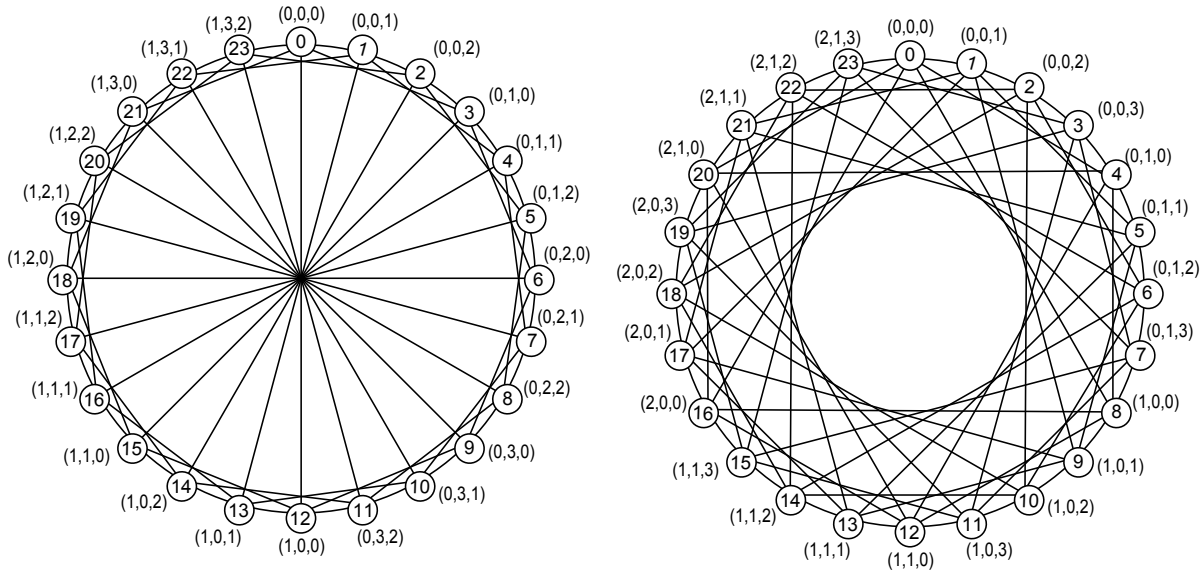

Figure 1: Two GRC graphs of 24 vertices, GR(2, 4, 3) (left) and GR(3, 2, 4) (right).

That is, $\mathrm{GR}(m_h, \ldots, m_2, m_1)$ can be partitioned into m_1 subgraphs isomorphic to $\mathrm{GR}(m_h, \ldots, m_2)$. Anyone of the subgraphs is induced exactly by those vertices with the same x_1 value. In addition, the edges not in the induced subgraphs form a cycle that links all vertices according to their numerical order. The partition process can be continued, and for all $i \leqslant h$, $\mathrm{GR}(m_h, m_{h-1}, \ldots, m_i)$ is still a GRC graph. Particularly, $\mathrm{GR}(m_h)$ is either a cycle of m_h vertices (if $m_h > 2$) or a 2-clique (if $m_h = 2$) [17]. For example, $\mathrm{GR}(2, 4, 3)$ shown in Figure 1(a) contains 3 disjoint copies of $\mathrm{GR}(2, 4)$, and $\mathrm{GR}(2, 4)$ contains four 2-cliques. Meanwhile, in Figure 1(b), $\mathrm{GR}(3, 2, 4)$ contains 4 disjoint copies of $\mathrm{GR}(3, 2)$, and $\mathrm{GR}(3, 2)$ contains two 3-cycles.

Two paths connecting two vertices in a simple graph is said to be *internally disjoint* if they have no common vertex except two end vertices. The *one-to-many routing* is to construct internally disjoint paths in a graph from a given vertex to each of the vertices in a given set that may contain all other vertices. One vertex can send copies of a message along internally disjoint paths to all other vertices in a graph to achieve fault-tolerant broadcasting [7, 14]. One vertex can also partition the message into multiple parts and send them separately to the destination vertices along internally disjoint paths to achieve secure message distribution [13].

A spanning tree of a graph is a tree that contains all vertices. Two spanning trees of a graph are *independent* if they are rooted at the same vertex r, and for every other vertex $v \neq r$, the two paths from r to v, one path in each tree, are internally disjoint. A set of spanning trees of a graph is said to be independent if they are pairwise independent. In 1989, Zehavi and Itai conjecture that, for any vertex r in a k-connected graph G, there exist k independent spanning trees of G rooted at r [25]. Although the conjecture has been proved for k-connected graphs with $k \leqslant 4$ ([7] for $k = 2$, [2, 25] for $k = 3$, and [3] for $k = 4$), it is still open for $k > 4$. Notice that constructing multiple independent spanning trees (IST for short) rooted at one vertex guarantees the one-to-many routing. As a result, lots of research results are presented for solving the IST problem in special graph classes (especially in interconnection networks), such as planar graphs [6, 11], chordal rings [8, 19], de Bruijn and Kautz graphs [4, 5], product graphs [1, 12], hypercubes [14, 16, 20], star graphs [15], folded hyper-stars [22], locally twisted cubes [10], recursive circulant graphs [21, 23], multidimensional tori [18], and folded hypercubes[24].

In this paper, we propose an algorithm to construct δ IST rooted at an arbitrary vertex of a GRC graph with degree δ. Particularly, the proposed algorithm is based on the multidimensional

label of a vertex and can be implemented easily and make the one-to-many routing parallelized.

The remaining part of this paper is organized as follows. Section 2 gives some essential notations and properties for the algorithm. Section 3 presents the algorithm. Section 4 proves the correctness of the algorithm. The last section contains our concluding remarks.

2. Preliminaries

Given a GRC graph $\mathrm{GR}(m_h, m_{h-1}, \ldots, m_1)$. Let $M_1 = 1$ and $M_i = m_{i-1} \cdot M_{i-1}$ for $i = 2, 3, \ldots, h$. The *ranking number* of vertex $x = (x_h, x_{h-1}, \ldots, x_1)$, denoted by $r(x)$, is the numerical order of x in decimal form. Based on the definition of GRC graphs, $r(x)$ can be obtained by using the following formula [17]:

$$r(x) \quad = \quad \sum_{i=1}^{h} x_i \cdot M_i.$$

For example, in Figure 1, the ranking number of each vertex of $\mathrm{GR}(2,4,3)$ and $\mathrm{GR}(3,2,4)$ is written in the cycle.

Proposition 2.1. *Suppose that vertex y is a neighbor of vertex x in a* GRC *graph. Then $|r(x) - r(y)| \equiv M_i \pmod{n}$ for $i \in \{1, 2, \ldots, h\}$.*

Proof. Since y is a neighbor of vertex x, by the definition of GRC graphs, $|y_h \cdots y_i \cdots y_1 - x_h \cdots x_i \cdots x_1| \equiv 1\overbrace{0\cdots0}^{i-1} \pmod{1\overbrace{0\cdots0}^{h}}$ in the mixed radix number system. That is, $|r(x) - r(y)| \equiv M_i \pmod{n}$. □

To explicitly represent the adjacency of vertices in a GRC graph, we say that vertex $x = (x_h, \ldots, x_i, \ldots, x_1)$ takes *jump* $i\pm$ to reach vertex $y = (y_h, \ldots, y_i, \ldots, y_1)$ if $x_j = y_j$ for $1 \leqslant j \leqslant i-1$ and number $y_h \cdots y_{i+1} y_i$ is equal to $x_h \cdots x_{i+1} x_i \pm 1$ in the mixed radix number system. Thus, the neighbors of x can be represented as a jump set $\{1-, 1+, 2-, 2+, \ldots, h-, h+\}$. In case of $m_h = 2$, jumps $h+$ and $h-$ reach the same vertex and they are viewed as one single jump $h-$.

The jump set representation is helpful to express the construction algorithm of IST. Similar to Propositions 2.1, we have the following proposition.

Proposition 2.2. *If vertex x takes jump $i+$ (respectively, $i-$) to reach a neighbor y, then $r(x) - r(y) = -M_i$ (respectively, M_i) for $i \in \{1, 2, \ldots, h\}$.*

Proof. When x takes jump $i+$ to reach y, $r(x) < r(y)$ (by taking modulo n). That is, $r(x) - r(y) = -M_i$. Conversely, $r(x) - r(y) = M_i$. □

Similar to the complement number of a number system, every vertex in a GRC graph has its complement vertex. For simplicity, every ranking number computation hereinafter is taken modulo n.

Definition 1. In a GRC graph, the *complement vertex* of vertex x is denoted by x' and $r(x') = n - r(x)$ where n is the number of vertices in the graph. In case $r(x)$=0 or $n/2$ (where n is even), x'=x.

Proposition 2.3. *If y is a neighbor of x, then y' is also a neighbor of x'.*

Proof. By Proposition 2.1, $|r(y) - r(x)| = M_i$ for $i \in \{1, 2, \ldots, h\}$. Then, $|r(y') - r(x')| = |(n - r(y)) - (n - r(x))| = M_i$. □

According to Propositions 2.3 and 2.2, we have the following lemma.

Lemma 2.4. *In a GRC graph, vertex x takes jump $i+$ (respectively, $i-$) to reach a neighbor y if and only if x' takes jump $i-$ (respectively, $i+$) to reach vertex y' for $i \in \{1, 2, \ldots, h\}$.*

Proof. When x takes jump $i+$ to reach y, by Proposition 2.2, $r(x) - r(y) = -M_i$. That is, $(n - r(x)) - (n - r(y)) = M_i$ or $r(x') - r(y') = M_i$. On the contrary, when x takes jump $i+$ to reach y, $r(x') - r(y') = -M_i$. □

Since GRC graphs are vertex-symmetric, without loss of generality, we can only consider the vertex $(0, 0, \ldots, 0)$ (also called vertex 0) as the root of IST on a GRC graph. Due to Lemma 2.4, when we determine a set of internally disjoint paths from vertex x to vertex 0, a set of internally disjoint paths from vertex x' to vertex 0 (self-complement) is also determined. Thus, only a half number of vertices need to be considered when solving the IST problem in a GRC graph.

Due to the internally disjoint requirement, it is obvious that the root has only one child in every independent spanning tree. If j is the jump taken by the only child for reaching the root, the independent spanning tree can be denoted by T_j. In Figure 2, for example, each of the IST rooted at vertex 0 in $GR(3,2,4)$ is named after the unique jump used to reach vertex 0.

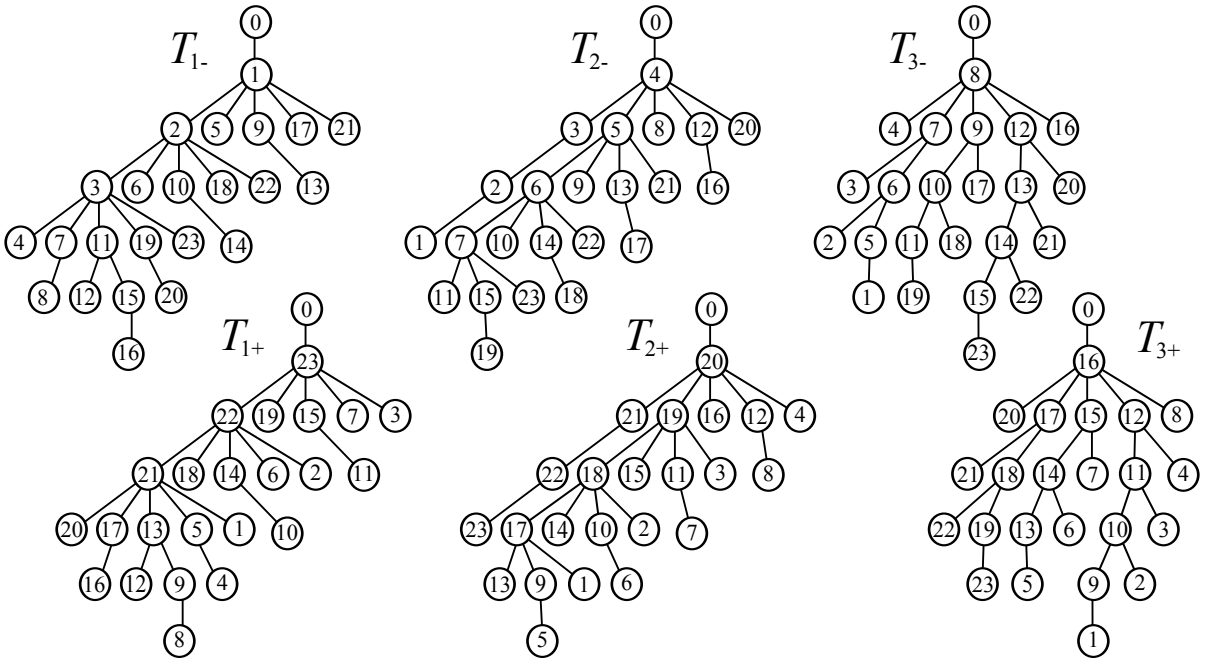

Figure 2: Six IST rooted at vertex 0 on GR(3, 2, 4).

For the IST on a GRCG, the jumps for reaching the root in all trees must be distinct. In addition, for a single non-root vertex, the jumps to reach its parents in different trees are also distinct. Since the jump sets are equivalent, the IST construction algorithm is simply to determine a bijection from the parent-reaching jump set to the root-reaching jump set for every vertex. For example, in Figure 2, vertex 7 reaches its parent by taking jumps $2-$, $1-$, $1+$, $3-$, $2+$ and $3+$ in the IST which are labeled by jumps $1-$, $2-$, $3-$, $1+$, $2+$ and $3+$, respectively.

When a jump j consecutively occurs k times ($k \geqslant 1$), we use $(j)^k$ to denote the jump sequence and call it a *leap*. A *leap sequence*, denoted by $[\ell_1, \ell_2, \cdots, \ell_t]$, is a sequence of t leaps ($t \geqslant 1$). Any path between two vertices in a GRC graph can be represented as a leap sequence.

Let $\dim(\ell)$ and $\mathrm{occ}(\ell)$ denote the dimension and the occurrences, respectively, of the jump that consists of leap ℓ. A leap sequence $[\ell_1, \ell_2, \cdots, \ell_t]$ is *strictly increasing* if for all $1 \leqslant i < j \leqslant t$, $\dim(\ell_i) < \dim(\ell_j)$. Further, a leap sequence is *regular* if it is strictly increasing and for each leap ℓ in the leap sequence, $\mathrm{occ}(\ell) < m_{\dim(\ell)}$.

The leap sequence $[\ell_s, \cdots, \ell_t, \ell_1, \cdots, \ell_{s-1}]$ is called a *rotation* of the leap sequence $[\ell_1, \cdots, \ell_{s-1}, \ell_s, \cdots, \ell_t]$ when $1 \leqslant s \leqslant t$. Then, we describe two special types of paths from vertex $x = (x_h, x_{h-1}, \ldots, x_1)$ to vertex 0 in a GRC graph by means of the rotations of a regular leap sequence. Note that we only consider half of the vertices, i.e., for vertex x where $r(x) \leqslant \lfloor n/2 \rfloor$.

Definition 2. An *α-path* from vertex x to vertex 0 is composed of a rotation of the regular leap sequence $\mathscr{L}$, where for each $x_i > 0$, there exactly exists a leap $(i-)^{x_i}$ in $\mathscr{L}$.

For example, see Figure 1(b), two rotations of the regular leap sequence $[(1-)^3, (2-)^1]$ form two α-paths from vertex $(0, 1, 3)$ to vertex 0 in GR(3, 2, 4). One α-path is the regular leap sequence itself, and another is $[(2-)^1, (1-)^3]$.

Definition 3. Let k be the left-most dimension where $x_k > 0$. Then, let $y = 1\overbrace{0\cdots0}^{k} - x_k \cdots x_1$ (in the mixed radix number system). A *β-path* from vertex x to vertex 0 is composed of a rotation of the regular leap sequence $\mathscr{L}$, where for each $y_i > 0$, there exactly exists a leap $(i+)^{y_i}$ in $\mathscr{L}$, and if $k < h$, a leap $((k+1)-)^1$ is especially added to the end of $\mathscr{L}$.

For example, see vertex $(0, 1, 3)$ in Figure 1(b) again, $y = (1, 0, 0) - (0, 1, 3) = (0, 0, 1)$ where

$k = 2$. Since $k < h$, the regular leap sequence $\mathscr{L} = [(1+)^1, (3-)^1]$. Two rotations of $\mathscr{L}$ form two β-paths from vertex $(0, 1, 3)$ to vertex 0 in GR$(3, 2, 4)$. Take vertex $(1, 0, 1)$ as an example of $k = h$, the regular leap sequence $\mathscr{L} = [(1+)^3, (2+)^1, (3+)^1]$. Three rotations of $\mathscr{L}$ form three β-paths from vertex $(1, 0, 1)$ to vertex 0 in GR$(3, 2, 4)$. They are $[(1+)^3, (2+)^1, (3+)^1]$, $[(2+)^1, (3+)^1, (1+)^3]$, and $[(3+)^1, (1+)^3, (2+)^1]$.

Accordingly, we have the following three lemmas:

Lemma 2.5. *In a GRC graph, any two α-paths from a vertex are internally disjoint.*

Proof. Suppose P_1 and P_2 are two different α-paths from vertex x to vertex 0. Let v_1 and v_2 be internal vertices in P_1 and P_2, respectively. If $v_1 = v_2$, then the two paths from x to v_1 and v_2 must contain the same leaps. Since the leap sequences of P_1 and P_2 are rotations of a regular leap sequence, this condition never happens. □

Lemma 2.6. *In a GRC graph, any two β-paths from a vertex are internally disjoint.*

Proof. The proof is similar to Lemma 2.5 and is thus omitted. □

Lemma 2.7. *In a GRC graph, an α-path and a β-path from a vertex are internally disjoint.*

Proof. Suppose P_1 be an α-path and P_2 be a β-path from vertex x to vertex 0. Let v_1 and v_2 be internal vertices in P_1 and P_2, respectively. If $v_1 = v_2$, then the two paths from x to v_1 and v_2 have the same leaps. Since the leap sequences of P_1 and P_2 have different signs in each dimension, this condition never happens. □

3. Constructing Independent Spanning Trees on a GRC graph

In this section, an algorithm is presented to construct IST rooted at vertex 0 in a GRC graph. The algorithm determines a bijection from the parent-reaching jump set to the root-reaching jump set for every non-root vertex. There are two phases in the proposed algorithm. In the first phase, for every non-root vertex, the jump set is partitioned into four jump sets. In the second phase, different rules are applied in each of the four jump sets to determine the bijection.

For each non-root vertex $x = (x_h, x_{h-1}, \ldots, x_1)$ in $GR(m_h, m_{h-1}, \ldots, m_1)$, the following procedure is performed to partition the entire jump set into four jump sets, i.e., A_x, B_x, C_x and D_x. Note that the procedure is designed for computing the first half of n vertices, i.e., $1 \leqslant r(x) \leqslant \lfloor n/2 \rfloor$.

Procedure JUMP_SET_PARTITION(x)
begin
1. $A_x = \emptyset$; $B_x = \emptyset$; $C_x = \emptyset$; $D_x = \emptyset$;
2. carry = 0;
3. Let k be the left-most dimension where $x_k > 0$.
4. **For** $i = 1$ to k **do**
5. **if** $(x_i > 0)$
6. $A_x = A_x \cup \{i-\}$;
7. **if** $((0 < x_i < m_i - 1)$ or $(x_i = 0$ and carry $= 1)$ or $(x_i = m_i - 1$ and carry $= 0))$
8. $B_x = B_x \cup \{i+\}$;
9. carry = 1;
10. **endif**
11. **if** $(x_i = 0$ and carry $= 0)$
12. $D_x = D_x \cup \{i-, i+\}$;
13. **if** $(x_i = 0$ and carry $= 1)$
14. $D_x = D_x \cup \{i-\}$;
15. **if** $(x_i = m_i - 1$ and carry $= 1)$
16. $D_x = D_x \cup \{i+\}$;
17. **enddo**
18. **if** $(k < h)$
19. $B_x = B_x \cup \{(k+1)-\}$;
20. **For** $j = k + 1$ to $h - 1$ **do**
21. $C_x = C_x \cup \{j+, (j+1)-\}$;
22. **if** $(m_h > 2)$
23. $D_x = D_x \cup \{h+\}$;
24. **endif**

end JUMP_SET_PARTITION

We use GR(3,2,4) as an example. Table 1 computes the four jump sets for the first half of vertices by using Procedure JUMP_SET_PARTITION.

Suppose all jumps in jump sets A_x and B_x are arranged in an increasing order of dimensions. We define the *predecessor jump* of a jump as its previous jump. For the first jump of the set, its predecessor jump is the last one of the set. In jump set C_x, jumps are added pairwise. Thus, each jump in one pair is the *partner jump* of the other jump.

When the jump set of vertex x is partitioned as mentioned above, the parents of x in every independent spanning tree can be determined by the following procedure.

Table 1: The four jump sets for vertex x ($1 \leqslant r(x) \leqslant 12$) in GR(3,2,4)

$r(x)$	x	A_x	B_x	C_x	D_x
1	(0,0,1)	$\{1-\}$	$\{1+, 2-\}$	$\{2+, 3-\}$	$\{3+\}$
2	(0,0,2)	$\{1-\}$	$\{1+, 2-\}$	$\{2+, 3-\}$	$\{3+\}$
3	(0,0,3)	$\{1-\}$	$\{1+, 2-\}$	$\{2+, 3-\}$	$\{3+\}$
4	(0,1,0)	$\{2-\}$	$\{2+, 3-\}$	$\emptyset$	$\{1-, 1+, 3+\}$
5	(0,1,1)	$\{1-, 2-\}$	$\{1+, 3-\}$	$\emptyset$	$\{2+, 3+\}$
6	(0,1,2)	$\{1-, 2-\}$	$\{1+, 3-\}$	$\emptyset$	$\{2+, 3+\}$
7	(0,1,3)	$\{1-, 2-\}$	$\{1+, 3-\}$	$\emptyset$	$\{2+, 3+\}$
8	(1,0,0)	$\{3-\}$	$\{3+\}$	$\emptyset$	$\{1-, 1+, 2-, 2+\}$
9	(1,0,1)	$\{1-, 3-\}$	$\{1+, 2+, 3+\}$	$\emptyset$	$\{2-\}$
10	(1,0,2)	$\{1-, 3-\}$	$\{1+, 2+, 3+\}$	$\emptyset$	$\{2-\}$
11	(1,0,3)	$\{1-, 3-\}$	$\{1+, 2+, 3+\}$	$\emptyset$	$\{2-\}$
12	(1,1,0)	$\{2-, 3-\}$	$\{2+, 3+\}$	$\emptyset$	$\{1-, 1+\}$

Algorithm PARENT_JUMPS_DETERMINE(x)
begin
Step 1. Determine A_x, B_x, C_x and D_x by using Procedure JUMP_SET_PARTITION(x);
Step 2. **for all** $j \in A_x$ **do**
Let jump p be the *predecessor jump* of jump j in A_x;
Vertex x takes jump j to reach its parent in tree T_p;
enddo
Step 3. **for all** $j \in B_x$ **do**
Let jump p be the *predecessor jump* of jump j in B_x;
Vertex x takes jump j to reach its parent in tree T_p;
enddo
Step 4. **for all** $j \in C_x$ **do**
Let jump t be the *partner jump* of jump j in C_x;
Vertex x takes jump j to reach its parent in tree T_t;
enddo
Step 5. **for all** $j \in D_x$ **do**
Vertex x takes jump j to reach its parent in tree T_j;
enddo
end PARENT_JUMPS_DETERMINE

We use GR(3,2,4) again as an example. Table 2 shows the parent-reaching jumps of the first half of vertices by using Algorithm PARENT_JUMPS_DETERMINE. According to the rules in the algorithm, a bijection from parent-reaching jump set to root-reaching jump set is established.

Table 2: The parent-reaching jumps of vertex x ($1 \leqslant r(x) \leqslant 12$) in different IST on GR(3,2,4)

$r(x)$	x	T_{1-}	T_{1+}	T_{2-}	T_{2+}	T_{3-}	T_{3+}
1	(0,0,1)	1−	2−	1+	3−	2+	3+
2	(0,0,2)	1−	2−	1+	3−	2+	3+
3	(0,0,3)	1−	2−	1+	3−	2+	3+
4	(0,1,0)	1−	1+	2−	3−	2+	3+
5	(0,1,1)	2−	3−	1−	2+	1+	3+
6	(0,1,2)	2−	3−	1−	2+	1+	3+
7	(0,1,3)	2−	3−	1−	2+	1+	3+
8	(1,0,0)	1−	1+	2−	2+	3−	3+
9	(1,0,1)	3−	2+	2−	3+	1−	1+
10	(1,0,2)	3−	2+	2−	3+	1−	1+
11	(1,0,3)	3−	2+	2−	3+	1−	1+
12	(1,1,0)	1−	1+	3−	3+	2−	2+

For constructing the IST shown in Figure 2, we have to transform the parent-reaching jumps in Table 2 to the ranking numbers of parent vertices. As shown in Table 3, the first half of the table ($1 \leqslant r(x) \leqslant 12$) are computed by using the formula of Proposition 2.2, while the second half of the table ($13 \leqslant r(x) \leqslant 23$) are obtained directly from the result of the first half by changing the sign of every jump (by Lemma 2.4).

4. Correctness Proof

To show the correctness of Algorithm PARENT_JUMPS_DETERMINE, we have to prove that the output of the algorithm are spanning trees of the input GRC graph. Then, we should prove that for every non-root vertex v, any two paths from v to the root in different spanning trees must be in-

Table 3: The parents of non-root vertex x in different IST on GR(3,2,4)

$r(x)$	x	T_{1-}	T_{1+}	T_{2-}	T_{2+}	T_{3-}	T_{3+}
1	(0,0,1)	0	21	2	17	5	9
2	(0,0,2)	1	22	3	18	6	10
3	(0,0,3)	2	23	4	19	7	11
4	(0,1,0)	3	5	0	20	8	12
5	(0,1,1)	1	21	4	9	6	13
6	(0,1,2)	2	22	5	10	7	14
7	(0,1,3)	3	23	6	11	8	15
8	(1,0,0)	7	9	4	12	0	16
9	(1,0,1)	1	13	5	17	8	10
10	(1,0,2)	2	14	6	18	9	11
11	(1,0,3)	3	15	7	19	10	12
12	(1,1,0)	11	13	4	20	8	16
13	(1,1,1)	9	21	5	17	12	14
14	(1,1,2)	10	22	6	18	13	15
15	(1,1,3)	11	23	7	19	14	16
16	(2,0,0)	15	17	12	20	8	0
17	(2,0,1)	1	21	13	18	9	16
18	(2,0,2)	2	22	14	19	10	17
19	(2,0,3)	3	23	15	20	11	18
20	(2,1,0)	19	21	4	0	12	16
21	(2,1,1)	1	22	5	20	13	17
22	(2,1,2)	2	23	6	21	14	18
23	(2,1,3)	3	0	7	22	15	19

ternally disjoint.

In Procedure Parent_Jumps_Determine, it turns out that every non-root vertex x takes different jumps to reach its parent in different spanning subgraphs (not yet proved to be spanning trees). To complete this proof, we need to show that there exists a unique path from every vertex x to 0 in each spanning subgraph. The following lemma depicts that the output of Algorithm Parent_Jumps_Determine are spanning trees.

Lemma 4.1. *Procedure* Parent_Jumps_Determine *can generate a set of δ spanning trees rooted at vertex 0 in a GRC graph, where δ is the degree or connectivity of the graph.*

Proof. According to the class of jump j of a non-root vertex x, we consider the following four cases:

Case 1: $j \in A_x$. x takes an α-path to the root.

Case 2: $j \in B_x$. x takes a β-path to the root.

Case 3: $j \in C_x$. x either takes one negative jump and then followed by a β-path to the root, or takes one positive leap and then followed by a β-path to the root.

Case 4: $j \in D_x$. x either takes one negative jump and then followed by an α-path to the root, or takes one positive jump and then followed by a β-path to the root.

Consequently, the resulting δ spanning subgraphs of Algorithm Parent_Jumps_Determine are spanning trees of the GRC graph. □

We now show the independency of the output IST of Algorithm Parent_Jumps_Determine.

Lemma 4.2. *All paths generated by jump sets A_x or B_x are internally disjoint.*

Proof. According to Lemmas 2.5, 2.6 and 2.7, all α-paths or β-paths are internally disjoint. □

Lemma 4.3. *All paths generated by jump set C_x or D_x are internally disjoint.*

Proof. In case a path generated by jump set C_x or D_x, x takes one jump and then takes either an α-path or a β-path to get to the root. We can prove that all the paths are internally disjoint by means of the summation of jumps. □

Lemma 4.4. *Let P and Q are two paths generated by jump sets $A_x \cup B_x$ and $C_x \cup D_x$, respectively. Then P and Q are internally disjoint.*

Proof. We can prove that P and Q are internally disjoint by means of the summation of jumps. □

According to Lemmas 4.1, 4.2, 4.3 and 4.4, we give the following theorem.

Theorem 1. *Algorithm* Parent_Jumps_Determine *can correctly construct δ IST on a GRC graph in $O(\delta N)$ time, where δ is the degree of a vertex, and N is the order of the graph.*

5. Concluding Remarks

In this paper, an algorithm is proposed to solve the IST problem on GRC graphs. Based on the algorithm, for each non-root vertex, a bijection from the parent-reaching jump set to root-reaching jump set is determined individually. Thus, the algorithm can be applied to the one-to-many parallel routing of a GRC graph.

References

[1] F. Bao, Y. Igarashi, and S. R. Öhring, Reliable broadcasting in product networks, *Discrete Applied Mathematics* 83 (1998) 3–20.

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

[3] S. Curran, O. Lee, and X. Yu, Finding four independent trees, *SIAM Journal on Computing*, 35 (2006) 1023–1058.

[4] Z. Ge and S. L. Hakimi, Disjoint rooted spanning trees with small depths in de Bruijn and Kautz graphs, *SIAM Journal on Computing* 26 (1997) 79–92.

[5] T. Hasunuma and H. Nagamochi, Independent spanning trees with small depths in iterated line digraphs, *Discrete Applied Mathematics* 110 (2001) 189–211.

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

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

[8] Y. Iwasaki, Y. Kajiwara, K. Obokata, and Y. Igarashi, Independent spanning trees of chordal rings, *Information Processing Letters* 69 (1999) 155–160.

[9] D. Knuth. *The Art of Computer Programming, Volume 2: Seminumerical Algorithms*, Third Edition, Addison-Wesley, 1997, pp.6566, 208209, and 290.

[10] J.-C. Lin, J.-S. Yang, C.-C. Hsu, J.-M. Chang, Independent spanning trees vs. edge-disjoint spanning trees in locally twisted cubes,*Information Processing Letters* 110 (2010) 414–419.

[11] K. Miura, D. Takahashi, S. Nakano, and T. Nishizeki, A linear-time algorithm to find four independent spanning trees in four-connected planar graphs, International Journal of Foundations of Computer Science 10 (1999) 195–210.

[12] K. Obokata, Y. Iwasaki, F. Bao, and Y. Igarashi, Independent spanning trees of product graphs and their construction, *IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences*, E79-A (1996) 1894–1903.

[13] M. O. Rabin, Efficient dispersal of information for secursity, load balancing, and fault tolerance, *Journal of the ACM* 36 (1989) 335–348.

[14] P. Ramanathan, K. G. Shin, Reliable broadcast in hypercube multicomputers, *IEEE Transactions on Computers* 37 (1988) 1654–1657.

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

[16] S.-M. Tang, Y.-L. Wang, and Y.-H. Leu, Optimal independent spanning trees on hypercubes, *Journal of Information Science and Engineering* 20 (2004) 605–617.

[17] S.-M. Tang, Y.-L. Wang and C.-Y. Li, Generalized recursive circulant graphs, *IEEE Transactions on Parallel and Distributed Systems*, 23 (2012) 87–93.

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

[19] J.-S. Yang, J.-M. Chang, S.-M. Tang, and Y.-L. Wang, Reducing the height of independent spanning trees in chordal rings, *IEEE Transactions on Parallel and Distributed Systems* 18 (2007) 644–657.

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

[21] 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 $d > 2$, *Theoretical Computer Science* 410 (2009) 2001–2010.

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

[23] J.-S. Yang, J.-M. Chang, S.-M. Tang, and Y.-L. Wang, Constructing multiple independent spanning trees on recursive circulant graphs $G(2^m, 2)$, *International Journal of Foundations of Computer Science* 21 (2010) 73–90.

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

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