This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2021.3080315, IEEE Access

IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Date of publication xxxx 00, 0000, date of current version xxxx 00, 0000.

Digital Object Identifier 10.1109/ACCESS.2017.DOI

Constructing Independent Spanning Trees on Generalized Recursive Circulant Graphs

DUN-WEI CHENG[1], KAI-HSUN YAO[2], AND SUN-YUAN HSIEH[3] (Senior Member, IEEE)

[1] Department of Computer Science and Information Engineering, National Cheng Kung University, No. 1, University Road, Tainan 701, TAIWAN. (e-mail: dunwei.ncku@gmail.com)
[2] Department of Computer Science and Information Engineering, National Cheng Kung University, No. 1, University Road, Tainan 701, TAIWAN. (e-mail: p76071226@gs.ncku.edu.tw)
[3] Department of Computer Science and Information Engineering, Institute of Medical Informatics and Center for Innovative FinTech Business Models, National Cheng Kung University, No. 1, University Road, Tainan 701, TAIWAN. (e-mail: hsiehsy@mail.ncku.edu.tw)

Corresponding author: Dun-Wei Cheng (e-mail: dunwei.ncku@gamil.com).

ABSTRACT The generalized recursive circulant networking can be widely used in the design and implementation of interconnection networks. It consists of a series of processors, each is connected through bidirectional, point-to-point communication channels to different neighbors. In this work, we apply the shortest path routing concept to build independent spanning trees on the generalized recursive circulant graphs. The proposed strategy loosen the restricted conditions in previous research and extended the result to a more general vertex setting by design the specific algorithm to deal with the constraint issue.

INDEX TERMS Independent Spanning Trees, Generalized Recursive Circulant Graphs, Interconnection Networks.

I. INTRODUCTION

INTERCONNECTION networks are used to be modeled by an undirected graph where a vertex represents a processor and an edge represents a communication channel. The spanning tree under a topology includes every vertex and connects without loops, massive applications [1]–[5] applied spanning tree structure to build efficient algorithms and solve related research problems. A set of spanning trees are said to be independent if they are rooted at the same root and for each of the remaining vertices there exists internally disjoint paths connect to the root. Considering an asynchronous system communicates by sending messages and data through an unreliable topology. The computation may not be able to conduct in a single processor due to the capacity issue and the distributed resources. To design reliable broadcasting, many researches [6], [7] construct independent spanning trees and restrict all message transmission through these structures. The results reveal that the communication complexity can be more efficient and improving the fault-tolerant ability. Secure information distribution protocols are desirable properties in data communication networks. Several researches [8], [9] have exploited the existence of disjoint structures to achieve efficient, reliable, and secure intentions. Sending different messages safely from the distributor to different destinations, an efficient algorithm to construct independent spanning trees can be applied to design a distribution protocol with high-level security requirements.

Circulant graphs [10]–[15] have a vast number of applications in communication routing [16], VLSI building [17], and distributed protocols [18]. Tang et al. [19] first proposed a superclass of recursive circulant graphs, generalized recursive circulant graphs (GRC graphs), which achieve more flexibility in the cardinality of the vertex set and construct by the circulant graph properties recursively. They investigate several properties of GRC graphs, such as proposed the shortest path routing algorithm and presented the diameter. In this paper, we proposed an efficient algorithm to construct independent spanning trees on GRC graphs. Previous research [20] proposed a method to construct independent spanning trees (ISTs) on a recursive circulant graph by the concept of shortest path routing. But the graph topology restricted under the base be greater than 2. Later, they [21] proposed a set of different rules to deal with the condition that every bases equal 2. In this work, we apply the shortest path routing concept and considering a more flexible setting that can conduct independent spanning trees on the GRC graphs but losing the restricted conditions on the base.

The rest of this paper is organized as follows: the proper-

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2021.3080315, IEEE Access

ties and notations of GRC graphs are introduced in Section 2. In Section 3, we present the proposed algorithm to construct independent spanning trees on GRC graphs. And Section 4 proves the correctness of our strategy and the experimental results on several complex GRC graph settings. The last section contains concluding remarks and future works.

II. PRELIMINARIES

GRC graphs are proposed by [19] with the recursive property as recursive circulant graphs, but a more general connection between vertices, which achieve more flexibility in the cardinality of the vertex set. GRC graphs are represented in a mixed radix number system, $GR(b_h, b_{h-1}, ...b_0)$, where $b_i \leq 2$ for $0 \leq i \leq h$. Index i is the position of this system and b_i refers to the base number (or radix) of corresponding position. For each vertex x labeled with $(x_h, x_{h-1}, ...x_0)$, where $0 \leq x_i < b_i$, presenting the label form of vertex x. Each vertex is linked to those vertices with only difference in one position by ± 1 of the mixed radix system. For instance, vertex $(1, 1, 2)$ in $GR(4, 2, 3)$ is adjacent to vertices $(1, 1, 1), (1, 1, 3), (1, 0, 2), (1, 2, 2), (0, 1, 2)$ and $(2, 1, 2)$. Note that carry and borrow mechanism in radix system are still implement. For instance, vertex $(1, 1, 2)$ in $GR(4, 2, 3)$ is adjacent to vertex$(1, 2, 2)$ by $+1$ in position 1. Since $x_1 = 2$ meets $b_1 = 2$, a carry should be added to the next position and x_1 should be reset to 0 at the same time. Therefore, the equivalence of $(1, 2, 2)$ and $(2, 0, 2)$ leads to the connection between vertices $(1, 1, 2)$ and $(2, 0, 2)$. As the structure of circulant graphs, the leftmost position will carry to the rightmost position and conduct the circulant property. Figure 1 shows the $GR(4, 2, 3)$.

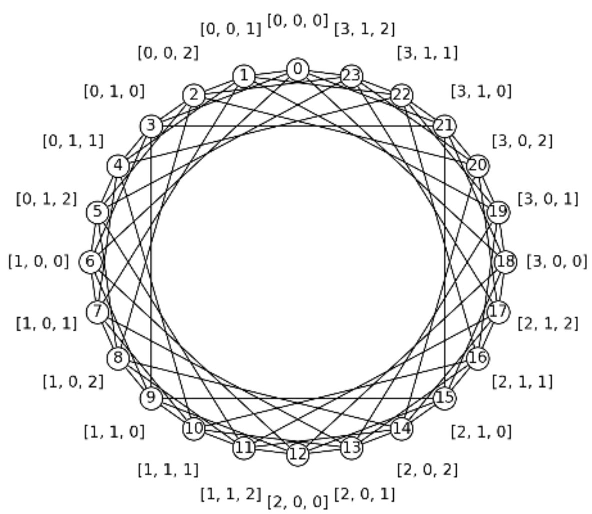

FIGURE 1. General Recursive Circulant Graph: $GR(4, 2, 3)$

Next, we elaborate the properties of GRC graphs and define the notations will be used in the proposed algorithms.

Property 1: GRC graphs are regular and vertex-symmetric.

Property 2: Given a GRC graph $GR(b_h, b_{h-1}, ...b_1, b_0)$. The degree δ_h of each vertex depends on the parameter h and the leftmost base b_h:

$$\delta_h = \begin{cases} 2h+2, & if\, b_h > 2; \\ 2h+1, & if\, b_h = 2. \end{cases}$$

Definition 1: Since of vertex-symmetric property of GRC graphs, without loss of generality, we choose the vertex $r = (0, 0, \ldots, 0, 0)$, where $x_i = 0, 0 \leq i \leq h$, to be the root vertex for each IST we built.

The connectivity of GRC graphs is δ_h for $h > 2$, we suggest that there exists δ_h ISTs according to the conjecture that the maximum number of ISTs is equal to the connectivity by previous study [22]. To specify which IST we are referring to thereafter, we give the definitions below:

Definition 2: We use i^+ (respectively, i^-) to represent a movement in the position i from vertex $(x_h, ..x_i, ..x_0)$ to vertex $(x_h, ..x_i + 1, ..x_0)$. (respectively, $(x_h, ..x_i - 1, ..x_0)$)

Definition 3: We denote the δ_h ISTs we built as $T_i^{\{+,-\}}$. If the last movement that reach to the root is i^+, we denote the IST as T_i^+. On the other hand, if the last movement is i^-, we denote the IST as T_i^-.

Example 1: In $GR(4, 2, 3)$, the parameter $h = 2$, so there are $\delta_h = 6$ ISTs in total. We denote as T_0^+, T_1^+, T_2^+, T_0^-, T_1^-, T_2^-.

Definition 4: We denote the paramter N as the cardinality of vertices in $GR(b_h, b_{h-1}, ..., b_1, b_0)$

$$N = \prod_{i=0}^{h} b_i$$

Definition 5: Given a sequence of movements, we take a set $M = \{m_0, \ldots, m_{|M|-1}\}$ to eliminate the identical movements. This set will present in ascending order according to the number of positions, we define the relation $succ(M, m_i) = m_{i+1}$ for $0 \leq i < |M| - 1$ and $succ(M, m_{|M|-1}) = m_0$.

III. CONSTRUCTING INDEPENDENT SPANNING TREES ON GRC GRAPHS

The overview of our proposed strategy to building independent spanning trees on a GRC graph is shown in Figure 2. According to the label form of each vertex, the proposed strategy will find the parent vertex in the specific spanning tree structure to reach the root vertex r. Our approach can be divided into three parts: the $SHORTEST\text{-}PATH$ algorithm, $AUGMENTED\text{-}PATH$ algorithm, and $FIND\text{-}PARENTS$ algorithm. The $FIND\text{-}PARENTS$ algorithm will return a $ParentTable$, that can be used to construct δ_h ISTs.

A. SHORTEST-PATH ALGORITHM

As mentioned in applications for building ISTs on interconnection networks, the major challenge tends toward reducing the heights of trees for better communication performance. Therefore, applying the shortest path concept proposed by

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2021.3080315, IEEE Access

Algorithm 1: MAIN(GR)

Input : $GR(b_h, b_{h-1}, \ldots, b_0)$, a GRC graph
Output: $ParentTable$

$i := 1$
while $i < N$ **do**
  $SP, D :=$ FIND_SHORTEST_PATH(GR, v_i)
  $APs :=$ FIND_AUGMENTED_PATH(SP, D, v_i)
  $ParentTable[v] :=$ FIND_PARENTS(SP, APs)
  $i := i + 1$
return $ParentTable$

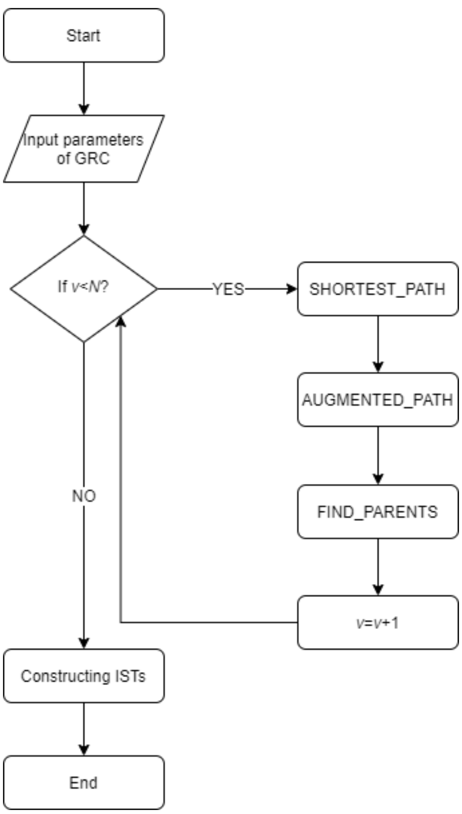

FIGURE 2. Workflow for constructing ISTs of GRC graphs

previous research [20] to construct ISTs on recursive circulant graphs. We follow the idea of a path-decomposition Latin square and according to the label form of each vertex to construct the shortest path to the root for distinct ISTs. A vertex takes a movement to its neighbor vertex in GRC graphs, the path decomposition means that a path can be expressed as a sequence of movements from the starting vertex. The length of a path is the number of movements it took, and the shortest path from a vertex to the root means it took the minimum movements.

For each base of a GRC graph, we consider the label form of v and make movements according to the distance between v and r. If the digit of position is bigger than half of the base we considering, the movement toward the base will be closer than it toward the 0. Therefore, we take movement i^+ in the shortest path, and vise versa. For instance, the vertex $v = (1, 1, 2)$ in $GR(4, 2, 3)$ will take the movement 0^+ for the rightmost position, since the digit of position is bigger than half of the base $x_0 = 2 > b_0/2 = 1.5$. In this example, one of the shortest paths will be $(1, 1, 2) \xrightarrow{0^+} (2, 0, 0) \xrightarrow{2^-} (1, 0, 0) \xrightarrow{2^-} (0, 0, 0)$ and the corresponding movements are $(0^+, 2^-, 2^-)$. We can rearrange the order of movements to get the other shortest paths.

Algorithm 2: SHORTEST-PATH(GR, v)

Input : $GR(b_h, b_{h-1}, \ldots, b_0)$, a GRC graph
$v = (x_h, \ldots, x_0)$, given vertex in label form
Output: SP, a set of shortest path movements.
D, a list of directions for each position.

$SP := \emptyset$; $D := []$; $i := 0$; $carry := false$
while $i <= h$ **do**
  if $carry$ is $true$ **then**
    $x_i := x_i + 1$
    $carry := false$
  if $x_i = 0$ **then**
    D.Append('$empty$')
  else if $x_i = b_i$ **then**
    D.Append('$full$')
    $carry := true$
  else if $i = h$ **then**
    if $b_h \geq 2x_h$ **then**
      $SP \cup \{i^-\}$
      D.Append('$down$')
    else
      $SP \cup \{i^+\}$
      D.Append('up')
  else if $b_i \geq 2x_i + 1$ **then**
    $SP \cup \{i^-\}$
    D.Append('$down$')
  else if $b_i = 2x_i$ ***and*** b_{i+1} is $even$ ***and*** $b_{i+1} > 2x_{i+1}$ **then**
    $SP \cup \{i^-\}$
    D.Append('$down$')
  else
    $k :=$ FIND_PIVOT(i, v, GR)
    if $k \neq null$ ***and*** $b_k \geq 2x_k + 2$ **then**
      $SP \cup \{i^-\}$
      D.Append('$down$')
    else
      $SP \cup \{i^+\}$
      D.Append('up')
      $carry := true$
  $i := i + 1$
return SP, D

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2021.3080315, IEEE Access

Algorithm 3: FIND_PIVOT(i, v, GR)

```
k := i
if x_k = b_k/2 then
    k := k + 1
    while k < h do
        if 2x_k + 1 = b_k then
            k := k + 1
        else
            return k
    return h
else
    return null
```

However, the shortest path concept proposed by previous research [20] could not be dealing with the situation for the base equals 2. When the base equals 2 in position i, the movements i^+ and i^- will conduct the identical results and this base-2 issue leading to conflicts when constructing ISTs. Therefore, previous research only constructed ISTs under restricted conditions that the base of the recursive graph needs to be greater than 2. In this work, we extend to the generalized recursive circulant graphs and loss this restricted condition. In the next section, we propose the $AUGMENTED\text{-}PATH$ algorithm to find the nearly shortest path for each vertex to connect to the root that would deal with the situation that base equals 2 on GRC graphs. But before we discuss further, we need to introduce a new concept named "directions" for the $SHORTEST\text{-}PATH$ algorithm. There are four directions {'up', '$down$', '$full$', '$empty$'} for each position based on the $SHORTEST\text{-}PATH$ algorithm.

- 'up'
  - This position's digit is counting upward to the base, and produces a carry to the next position.
- '$down$'
  - This position's digit is counting downward to '0'.
- '$full$'
  - This position's digit meets the base after adding carry from former position, then also produces a carry to the next position.
- '$empty$'
  - This position's digit is '0'.

According to the label form of vertex v, the modified $SHORTEST\text{-}PATH$ outputs a set SP conduct from a sequence of movements through this shortest path. We also record a list of directions D for each position which will be used to construct ISTs for the rest part of our strategy. For instance, the vertex $v = (1, 1, 2)$ in $GR(4, 2, 3)$, this algorithm outputs $SP = \{0^+, 2^-\}$ and $D =$ ['$down$', '$full$', 'up'] for each corresponding position.

B. AUGMENTED-PATH ALGORITHM

We propose the $AUGMENTED\text{-}PATH$ algorithm to find a nearly shortest path for each vertex to connect to the root that would deal with the situation that base equals 2 on GRC graphs. We conduct several patterns to construct augmented path AP, which is similar to SP but the premise is that AP will not take the same movement in SP.

Algorithm 4: AUGMENTED-PATH(SP, D, v)

Input : SP, a set of shortest path movements.
D, a list of directions for each position.
$v = (x_h, \dots, x_0)$, given vertex in label form

Output: APs, augmented paths we constructed.

```
APs, AP := ∅, ∅
i, carry := 0, false
while i < h do
    if carry is true then carry := false,
        x_i := x_i + 1
    if x_i = b_i then carry := true
    else
        if AP.isEmpty() then
            // meets half property
            if x_i = b_i/2 then
                if i^- ∈ SP then
                    AP.Append(i^+)
                    carry := true
                else if i^+ ∈ SP then
                    AP.Append(i^-)
            else if D[i] is 'up' then
                carry := true
        else
            // pattern matching
            if b_i = 2 and b_{i+1} = 2 then
                Let p be x_{i+1}(D[i+1])x_i(D[i])
            else if b_i = 2 and b_{i+1} ≠ 2 then
                Let p be x_{i+1}(D[i+1])x_i(D[i])
                APs, AP, carry :=
                    PATTERN_MATCH_SECOND(p)
            else if b_i ≠ 2 then
                Let p be x_i(D[i])
                APs, AP, carry :=
                    PATTERN_MATCH_THIRD(p)
    i := i + 1
return APs
```

From the shortest path concept, we observe a "half-property" for GRC graphs. When the digit of the position is exactly half of the base ($x_i = v_i/2$), the distance toward the base is equal to the distance toward the 0. Therefore, in that position, we took an opposite direction to conduct an augmenting path with the same distance to reach the 0. In the $AUGMENTED\text{-}PATH$ algorithm, we make an opposite movement when the position satisfying the half-property. Leading the augmented path to take distinct movements with SP. Note that this algorithm will match the appropriate

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2021.3080315, IEEE Access

pattern according to the digit and direction of each position, and the digit will change by the $carry$ from the previous movement.

Algorithm 5: PATTERN_MATCH_FIRST(p)

Input : p, the pattern $x_{i+1}(D[i+1])x_i(D[i])$
Output: APs,AP,$carry$
begin
  if p *is* $1(up)1(empty)$ **then**
    /* case 1:counting up to connect to SP with the same length */
    $APs, AP :=$ APPENDCLEAR$(APs, AP, i, +)$
  else if p *is* $0(down)1(full)$ **then**
    /* case 1′:counting down to connect to SP with the same length */
    $APs, AP :=$ APPENDCLEAR$(APs, AP, i, -)$
    $carry := true$
  else if p *is* $1(full)1(full)$ **then**
    /* case 2:counting up to connect to SP/AP with length+1, then meet another half property */
    $APs, AP :=$ APPENDCLEAR$(APs, AP, i, +)$
    $AP.Append(i^-)$
  else if p *is* $0(empty)1(empty)$ **then**
    /* case 2′:counting down to connect to SP/AP with length+1, then meet another half property */
    $APs, AP :=$ APPENDCLEAR$(APs, AP, i, -)$
    $AP.Append(i^+)$
    $carry := true$
  else if p *is* $1(down)1(empty)$ **then**
    /* case 3:counting up and not connect to SP */
    $AP.Append(i^+)$
    $carry := true$
  else if p *is* $0(up)1(full)$ **then**
    /* case 3′:counting down and not connect to SP */
    $AP.Append(i^-)$
  return APs,AP,$carry$

When designing the $AUGMENTED\text{-}PATH$ algorithm, we apply the symmetric property of GRC graphs to construct the augmented paths. The symmetric property will lead to a mirror case for each pattern, that is the directions are corresponding from up to $down$ (vice versa) and from $empty$ to $full$ (vise versa). Illustrating the algorithm, we find the first position that satisfies the half property and the base of the next two positions both equals 2. Assuming the position that satisfies the half property took i^- movement in SP. Then according to the half property, the augmented path will take the opposite movement i^+ and conduct a $carry$ for the next position. When the next two positions of vertex v match the pattern $1(up)1(empty)$, which is case 1 in this algorithm. Since the base of these two positions both equals 2, the augmented path will take an upward movement $(+)$ and conduct the same result as the $SHORTEST\text{-}PATH$ algorithm with the same length. There will be a mirror case assuming the position that satisfies the half property took i^+ movement in SP. Then according to the half property, the augmented path will take the opposite movement i^-. When the next two positions of vertex v match the pattern $0(down)1(full)$, which is case 1' in this algorithm. Since the base of these two positions both equals 2, the augmented path will take a downward movement $(-)$ and conduct the same result as the $SHORTEST\text{-}PATH$ algorithm with the same length.

In the $AUGMENTED\text{-}PATH$ algorithm, we divide each position of the GRC graphs into two groups according to the base value. When the base value equals 2, we assign this position into the "2-factor group." And besides those positions, we also assign one left more position into the "2-factor group." The rest of the positions will be assigned to the "N-factor group." For instance, $GR(4,8,6,2,2,2,5)$, the 2-factor group will be $\{1,2,3,4\}(b_1 = 2, b_2 = 2, b_3 = 2, b_4 = 6)$ and the N-factor group will be $\{0,5,6\}(b_0 = 5, b_5 = 8, b_6 = 4)$. According to the half property and the movements in SP, the $AUGMENTED\text{-}PATH$ algorithm constructs augmented paths to connect to the root with the nearly shortest path. We divided the bases of the GRC graph into two groups and concluded several patterns and their mirror case to construct those paths. Each augmented path will take distinct movement based on the digit and direction in the corresponding positions. For instance, the vertex $v = (0,1,1,0,1)$ in $GR(8,2,2,2,2)$ will conduct $SP = \{0^-, 2^+, 4^-\}$ and $D = [\text{'}down\text{'}, \text{'}full\text{'}, \text{'}up\text{'}, \text{'}empty\text{'}, \text{'}down\text{'}]$. At position 0, which satisfies the half property. The $SHORTEST\text{-}PATH$ algorithm took the movement 0^-. Therefore, the $AUGMENTED\text{-}PATH$ algorithm takes the opposite movement 0^+ and connects to vertex $v' = (0,1,1,1,0)$ by adding $carry$ from previous position. The next two positions match the pattern for case 1 and taking the movement 1^+. The $AP = \{0^+, 1^+\}$ and we keep checking the next position for another augmented path. At position 2, which also satisfies the half property. The $SHORTEST\text{-}PATH$ algorithm took the movement 2^+. Therefore, the $AUGMENTED\text{-}PATH$ algorithm takes the opposite movement 2^- and connects to vertex $v'' = (0,1,0,0,0)$. The next two positions match the pattern for case 1' and taking the movement 3^-. In this instance the $APs = \{\{0^+, 1^+\}, \{2^-, 3^-\}\}$.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2021.3080315, IEEE Access

Algorithm 6: PATTERN_MATCH_SECOND(p)

```
Input  : p, the pattern x_{i+1}(D[i+1])x_i(D[i])
Output: APs,AP,carry
begin
    // X:an arbitrary digit
    else if p is X(up)1(empty) then
        /* case 4:similar to case 1   */
        APs, AP :=
         AppendClear(APs, AP, i, +)
    if p is X(down)1(full) then
        /* case 4':similar to case 1'
            */
        APs, AP :=
         AppendClear(APs, AP, i, −)
    else if p is X(full)1(full) then
        /* case 5:similar to case 2   */
        APs, AP :=
         AppendClear(APs, AP, i, +)
        AP.Append(i^−)
    else if p is 0(empty)1(empty) then
        /* case 5':similar to case 2'
            */
        APs, AP :=
         AppendClear(APs, AP, i, −)
        AP.Append(i^+)
    else if p is X(down)1(empty) then
        /* case 6:counting down and
            connect to SP, then meet
            another half property     */
        APs, AP :=
         AppendClear(APs, AP, i, −)
        AP.Append(i^+)
    else if p is X(up)1(full) then
        /* case 6':counting up and
            connect to SP, then meet
            another half property     */
        APs, AP :=
         AppendClear(APs, AP, i, +)
        AP.Append(i^−)
    return APs,AP,carry
```

Algorithm 7: PATTERN_MATCH_THIRD(p)

```
Input  : p, the pattern x_i(D[i])
Output: APs,AP,carry
begin
    if p is 1(empty) then
        /* case 7:counting down to
            connect to SP/AP with
            lenght+1                    */
        APs, AP :=
         AppendClear(APs, AP, i, −)
    else if p is X(full) and X = b_i − 1 then
        /* case 7':counting up to
            connect to SP/AP with
            lenght+1                    */
        APs, AP, carry :=
         AppendClear(APs, AP, i, +);
        carry := true
    else if p is X(down) then
        if AP.Length() = 1 then
            AP.Append(i^+)
            if X < ⌊b_i/2⌋ then
                APs.Append(AP); AP.Clear()
            else if X = ⌊b_i/2⌋ then
                carry := true
        else if AP.Length() > 1 then
            APs.Append(AP); AP.Clear()
    else if p is X(up) then
        if AP.Length() = 1 then
            APs, AP, carry :=
             AppendClear(APs, AP, i, −)
            carry := true
        else if AP.Length() > 1 then
            APs.Append(AP)
            AP.Clear(); carry := true
        if i ∈ N − factor then
            if x_i = b_i/2 then
                APs, AP, carry :=
                 AppendClear(APs, AP, i, −)
                carry := true
            else if p is 1(empty) then
                APs, AP, carry :=
                 AppendClear(APs, AP, i, −)
            else if p is N − 1(full) then
                APs, AP, carry :=
                 AppendClear(APs, AP, i, +)
                carry := true
            else
                AP.Clear()
    return APs,AP,carry
```

C. FIND-PARENTS

The $FIND\text{-}PARENT$ algorithm conducts the δ_h distinct parents for each vertex and returns the results to form the $ParentTable$ for constructing the δ_h ISTs. The previous two algorithms conduct the movement set SP and APs for each vertex in the GRC graphs. In this section, we apply the subroutine $CLASSIFIER$ to find four variation paths according to the previous movement results. The variations SP^* and AP^* will be the extension of the previous movement results; the variation $\overline{SP}$ takes the opposite movement in the "N-factor" position; and the last variation $others$

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2021.3080315, IEEE Access

Algorithm 8: APPENDCLEAR(APs, AP, i, s)

Input : APs, AP
i, the position.
$s \in \{+, -\}$, denote counting up/down.
output: APs,AP
begin
  $AP.Append(i^s)$
  $APs.Append(AP)$
  $AP.Clear()$
  return APs, AP

conduct movements that belong to this GRC graph but not have been used in the previous situations. Here, we use the notation $\bigoplus$ to represent that the vertex v takes the movement m and the opposite of movement m can be denoted as $-m$. For instance, the vertex $v = (0, 1, 1, 0, 1)$ in $GR(8, 2, 2, 2, 2)$ will conduct $SP = \{0^-, 2^+, 4^-\}$ and $APs = \{\{0^+, 1^+\}, \{2^-, 3^-\}\}$. The $parents\_for\_ISTs =$ $[(1, 0, 0, 0, 1), (0, 1, 0, 1, 1), (0, 0, 1, 0, 1), (0, 1, 0, 0, 1),$ $(0, 1, 1, 0, 0), (1, 1, 1, 0, 1), (1, 0, 1, 0, 1), (7, 1, 1, 0, 1),$ $(0, 1, 1, 1, 0), (0, 1, 1, 1, 1)]$. The index of $parents\_for\_IST$ from 0 to $\delta_h - 1$ are referring to IST and list below in order. $T_0^-, T_1^-, T_2^-, T_3^-, T_4^-, T_4^+, T_3^+, T_2^+, T_1^+, T_0^+$.

Algorithm 9: FIND_PARENTS(SP, APs)

Input : SP, APs
Output: $parents\_for\_ISTs$
$parents\_for\_ISTs := []$
$SP^*, AP^*, \overline{SP}, others, dict :=$ CLASSIFIER(SP, APs, δ_h)
while $i < \delta_h$ **do**
  $m := dict[i]$
  if *m is a movement exists in AP* **then**
    $parents\_for\_ISTs[i] := v \bigoplus succ(AP, m)$
  else if $m \in SP$ **then**
    $parents\_for\_ISTs[i] := v \bigoplus succ(SP, m)$
  else if $m \in \overline{SP} \cup SP^* \cup AP^*$ **then**
    $parents\_for\_ISTs[i] := v \bigoplus m$
  else
    $parents\_for\_ISTs[i] := v \bigoplus m$
  $i := i + 1$
return $parents\_for\_ISTs$

IV. CORRECTNESS

Lemma 1: Each subgraph we constructed is a spanning tree on the GRC graph.

Proof: First, we find a parent vertex for each vertex in distinct δ_h subgraphs we constructed. Since this algorithm iterates each vertex in the GRC graphs once. It guarantees the δ_h spanning subgraphs on the GRC graph. Then, we prove that this spanning subgraph is a tree structure. There is only

Algorithm 10: CLASSIFIER(SP, APs, δ_h)

Input : SP, APs, δ_h
Output: $SP^*, AP^*, \overline{SP}, others, dict$
$i := 0$
$sign := null$
$dict := null$
while $i < \delta_h$ **do**
  if $2 * i \leq \delta_h - 1$ **then**
    $sign := -$
    $u := i$
  else
    $sign := +$
    $u := \delta_h - 1 - i$
  $m := u^{sign}$
  $dict[i] := m$
  if $m \notin (AP \cup SP)$ **then**
    if $u \in 2-factor$ ***and*** $(u + 1)^{\{+,-\}} \in SP$ **then**
      $SP^*.Append(m)$
    else if $u \in 2-factor$ ***and*** $(u + 1)^{\{+,-\}} \in AP$ **then**
      $AP^*.Append(m)$
    else if $u \in N-factor$ ***and*** $-m \in SP$ **then**
      $\overline{SP}.Append(m)$
    else
      $others.Append(-m)$
  $i := i + 1$
return $SP^*, AP^*, \overline{SP}, others, dict$

one path in each subgraph we constructed that connects this vertex to the root. Considering the following cases, we denote T_m as the IST that the last movement connects to the root is m:

Case 1: Suppose $SP = \{m_0, m_1, \ldots, m_{t-1}\}$ and $m_i \in SP$, we have its decomposition with $(m_{i+1}, m_{i+2}, \ldots, m_i)$ for T_{m_i}. By $FIND\text{-}PARENT$, we can get the vertex's parent v_{p1} after taking m_{i+1} and the length of decomposition will minus one. As for vertex v_{p1}, its decomposition become $(m_{i+2}, \ldots, m_i)$. In the end, with the same operation, the vertex can be routed to the child vertex of root with decomposition (m_i) then connect to root after taking the movement.

Case 2: Suppose $SP = \{m_0, m_1, \ldots, m_{t-1}\}$ and $m_i \in others$. The vertex will take $-m_i$ as the first movement to its parent v_{p1} in T_{m_i} by $FIND\text{-}PARENT$. Then v_{p1}'s SP must contain m_i and its decomposition is $(m_{i+1}, \ldots, m_i)$. By taking the movements in order like Case 1, the vertex will be routed to the child vertex of root with decomposition (m_i) then connect to root after taking the

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2021.3080315, IEEE Access

movement.

Case 3: Suppose $SP = \{m_0, m_1, \ldots, m_{t-1}\}$ and $m_i \in \overline{SP}$, that is, $-m_i$ in SP. By $FIND\text{-}PARENTS$, the vertex will take m_i at first to its parent v_{p1} in T_{m_i}. If m_i still $\notin SP$ of v_{p1}, the vertex will keep taking the movement m_i until to the ancestor whose SP contain m_i. Then like Case 1, the decomposition becomes $(\ldots, m_i)$ with m_i be the last movement. In the end, the vertex will be route to the child vertex of root with decomposition (m_i) then connect to root after taking the movement.

Case 4: Suppose $m_i \in AP, AP = \{\ldots, m_i, m_{i+1} \ldots, \}$. The vertex will take m_{i+1} to its parent v_{p1} in T_{m_i} by $FIND\text{-}PARENT$. At v_{p1}, either $m_i \in SP$ or $m_i \in AP$. If $m_i \in SP$, like Case 1, the decomposition will be $(\ldots, m_i)$. In the end, the vertex will be routed to the child vertex of root with decomposition (m_i) then connect to root after taking the movement. If $m_i \in AP$, using the same way to its parent v_{p2} after taking movement by $FIND\text{-}PARENT$. Eventually, m_i will belong to SP at one ancestor. In the end, the vertex will be routed to the child vertex of root with decomposition (m_i) then connect to root after taking the movement.

Case 5: Suppose $m_i \in SP^*$ and m_i's position is i, Because the movement at position $i+1 \in SP$. If $D[i+1] =$ 'up', by taking m_i and routed to its parent result in x_i from 0 to 1, and it will make $m_i \in SP$. Otherwise, if $D[i+1] =$ '$down$', the vertex also is routed to parent whose SP contain m_i. Like Case 1, the decomposition becomes $(\ldots, m_i)$ with mv_i be the last movement. In the end, the vertex will be routed to the child vertex with decomposition (m_i) then connect to root after taking the movement.

Case 6: Suppose $m_i \in AP^*$ and m_i's position is i, Because there's a movement at position $i+1 \in AP$. By taking m_i and routed to its parent in T_{m_i}, which result in x_i from 0 to 1, and it will make $m_i \in AP$. Like Case 4, eventually the routed path will reach to the ancestor that contain m_i in its SP. In the end, the vertex will be routed to the child vertex of root with decomposition (m_i) then connect to root after taking the movement.

Lemma 2: According to the algorithm we proposed, if there are two paths from one vertex to the root and the first movement of one path would not exist in the movements of the other path, then these two paths are internally disjoint.

Proof: Let the decompositions of these two paths be $P_a = (m_{a1}, m_{a2} \ldots m_{at})$ and $P_b = (m_{b1}, m_{b2} \ldots m_{bn})$. The vertex set Z_a and Z_b denote the vertices that each path passed. These two paths starting from the same vertex to root are internally disjoint if and only if the vertices they passed can't be equivalent. That is, $\forall x \in Z_a, \forall y \in Z_b, x \not\equiv y$. Due to m_{a1} as the first movement in P_a, we know that v must took m_{a1} to x. However, since $m_{a1} \notin P_b$, if the combination of m_{a1} with other movements in P_a could not equivalent to movements in P_b, then $x \not\equiv y$. If the combination of m_{a1} with other movements in P_a could equivalent to movements in P_b, because there must be one movement in the combination as the last movement in P_a, from definition of Z_a, combination can't be met, so $x \not\equiv y$.

TABLE 1. experiment results

GRC graph	N	δ_h	Height Max.	Height Avg.	Run time(ms)
$GR(4,2)$	8	4	4	2.125	0.4776
$GR(4,2,2)$	16	6	5	2.625	1.2317
$GR(4,2,2,2)$	32	8	6	3.055	1.4821
$GR(4,2,2,2,2)$	64	10	7	3.456	3.4754
$GR(4,2,2,2,2,2)$	128	12	8	3.850	10.3512
$GR(4,2,2,2,2,2,2)$	256	14	10	4.616	97.9597
$GR(3,3)$	9	4	3	2.111	0.607
$GR(3,3,3)$	27	6	4	2.926	2.1377
$GR(3,3,3,3)$	81	8	5	3.642	7.3423
$GR(3,3,3,3,3)$	243	10	6	4.325	24.8076
$GR(10,10)$	100	4	14	7.175	2.7766
$GR(10,10,10)$	1000	6	19	9.65	41.4834
$GR(10,10,10,10)$	10000	8	23	12.108	507.5772
$GR(4,2,3)$	24	6	5	2.917	1.6722
$GR(3,4,5)$	60	6	7	3.867	4.7639
$GR(6,7,8,9)$	6024	8	17	9.059	160.5518
$GR(10,11,12,13)$	17160	8	27	13.990	1050.1259
$GR(6,2,2,5)$	120	8	9	4.65	5.3769
$GR(43,19,27)$	22059	6	64	29.223	1080.1769

TABLE 2. Experimental environment

	Specific configuration
OS	macOS High Sierra
CPU	Intel Core i5 2.3GHz
RAM	8GB
Programming language	Python 3.7.3

Theorem 1: According to the algorithm we proposed, we construct δ_h independent spanning trees on the generalized recursive circulant graphs.

Proof: Given a vertex, from Lemma 1 we can construct each path to root for all spanning trees. And by Lemma 2, it can prove all these paths are pairwise internally disjoint. Therefore, the δ_h spanning trees we constructed are independent to each other.

In the $SHORTEST\text{-}PATH$ algorithm, it takes $O(h)$ time by iterating every position in GRC. And in the $AUGMENTED\text{-}PATH$ algorithm, it also takes $O(h)$ to construct all augmented path for the same reason. In the $FIND\text{-}PARENT$ algorithm, it decides parents through every i, where $0 \leq i < \delta_h$. In addition, δ_h is actually $O(h)$. Therefore, it takes $O(h)$ time in the $FIND\text{-}PARENT$ algorithm. Lastly, since all vertices have to do all the procedures above. The aggregate of time in this strategy to construct δ_h ISTs in a GRC graph takes $O(Nh)$.

We also discuss the experimental results that implementing the proposed strategy to construct δ_h ISTs on the various GRC graphs. Table IV reveals that the cardinality of the GRC graphs we experimenting with from the simplest 8 vertices to 22,059 vertices. The running time of each experiment took under one second, the experiment environment is showing

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2021.3080315, IEEE Access

in Table IV, and most simple GRC graphs took less than one nanosecond to construct δ_h ISTs. Note that δ_h is the maximum number of ISTs that a GRC graph can construct. And in Table IV, we count the height information of each ISTs and remark the average height and the maximal height we built. As mention before, in applications for building ISTs on interconnection networks, the major challenge tends toward reducing the heights of trees for better communication performance. The experiment results reveal that the average heights of ISTs we constructed on complex GRC graphs (the cardinality of the vertex set is higher than 10,000 vertices) is less than 30 levels. Figure 3, Figure 4, and Figure5 are the $\delta_h = 6$ ISTs constructed according to our proposed algorithm on $GR(4, 2, 3)$. Since in $GR(4, 2, 3)$, the cardinality of the vertex set is 24, we label each vertex from 0 to 23.

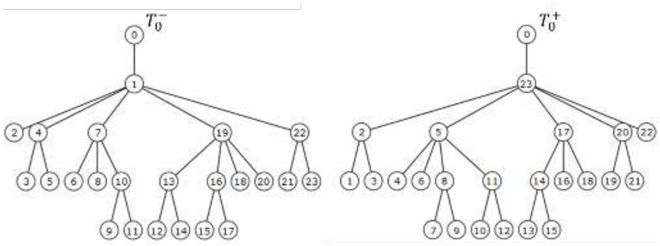

FIGURE 3. ISTs T_0^- and T_0^+ we constructed on $GR(4, 2, 3)$

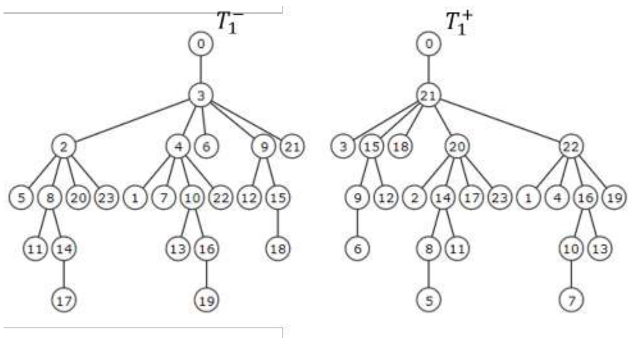

FIGURE 4. ISTs T_1^- and T_1^+ we constructed on $GR(4, 2, 3)$

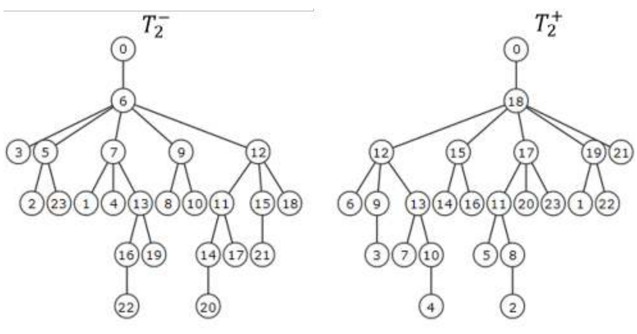

FIGURE 5. ISTs T_2^- and T_2^+ we constructed on $GR(4, 2, 3)$

V. CONCLUSION

In this work, we apply the shortest path routing concept to build independent spanning trees on the generalized recursive circulant graphs. The proposed strategy loosen the restricted conditions in previous research and extended the result to a more general vertex setting by design a specific algorithm to deal with the constraint issue. The GRC graphs can be widely used in the implementation of interconnection networks, massive applications applied spanning tree structure to build efficient algorithms and solve related applications such as reliable broadcasting and secure distributed protocols. The major challenge tends toward reducing the heights of trees for better communication performance. According to the label form of each vertex, our proposed strategy follows the shortest path routing concept and finds a parent vertex in the specific spanning tree structure to reach the root. We also propose the $AUGMENTED\text{-}PATH$ algorithm to find the nearly shortest path for each vertex to connect to the root that would deal with the situation that base equals 2 on GRC graphs.

The aggregate of time in this strategy to construct δ_h ISTs in a GRC graph takes $O(Nh)$ and satisfying the conjecture that the connectivity is equal to the number of vertex-disjoint spanning trees. We discuss the experimental results that implementing the proposed strategy to construct δ_h ISTs on the various GRC graphs. The cardinality of the GRC graphs we experimenting with from the simplest 8 vertices to 22,059 vertices and the running time of each experiment took under one second. The experiment results also reveal that the average heights of ISTs we constructed on complex GRC graphs. When the cardinality of the vertex set is higher than 10,000 vertices, the average height of δ_h ISTs is less than 30 levels. We will consider the optimal height of ISTs as our future work, to conduct a more efficient algorithm for the GRC related graph structures.

REFERENCES

[1] J.-F. Huang, S.-S. Kao, S.-Y. Hsieh, and R. Klasing, “Top-down construction of independent spanning trees in alternating group networks,” IEEE Access, vol. 8, pp. 112 333–112 347, 2020.

[2] C.-F. Lin, J.-F. Huang, and S.-Y. Hsieh, “Constructing independent spanning trees on transposition networks,” IEEE Access, vol. 8, pp. 147 122–147 132, 2020.

[3] D.-W. Cheng, C.-T. Chan, and S.-Y. Hsieh, “Constructing independent spanning trees on pancake networks,” IEEE Access, vol. 8, pp. 3427–3433, 2019.

[4] B. Cheng, D. Wang, and J. Fan, “Constructing completely independent spanning trees in crossed cubes,” Discrete Applied Mathematics, vol. 219, pp. 100–109, 2017.

[5] S.-S. Kao, K.-J. Pai, S.-Y. Hsieh, R.-Y. Wu, and J.-M. Chang, “Amortized efficiency of constructing multiple independent spanning trees on bubble-sort networks,” Journal of Combinatorial Optimization, vol. 38, no. 3, pp. 972–986, 2019.

[6] A. Itai and M. Rodeh, “The multi-tree approach to reliability in distributed networks,” Information and Computation, vol. 79, no. 1, pp. 43–59, 1988.

[7] F. Bao, Y. Funyu, Y. Hamada, and Y. Igarashi, “Reliable broadcasting and secure distributing in channel networks,” IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences, vol. 81, no. 5, pp. 796–806, 1998.

[8] A. A. Rescigno, “Vertex-disjoint spanning trees of the star network with applications to fault-tolerance and security,” Information Sciences, vol. 137, no. 1-4, pp. 259–276, 2001.

[9] J.-S. Yang, H.-C. Chan, and J.-M. Chang, "Broadcasting secure messages via optimal independent spanning trees in folded hypercubes," Discrete Applied Mathematics, vol. 159, no. 12, pp. 1254–1263, 2011.
[10] B. Alspach, S. C. Locke, and D. Witte, "The hamilton spaces of cayley graphs on abelian groups," Discrete mathematics, vol. 82, no. 2, pp. 113–126, 1990.
[11] F. T. Boesch and A. P. Felzer, "A general class of invulnerable graphs," Networks, vol. 2, no. 3, pp. 261–283, 1972.
[12] F. Boesch and R. Tindell, "Circulants and their connectivities," Journal of Graph Theory, vol. 8, no. 4, pp. 487–499, 1984.
[13] R. C. Entringer, D. E. Jackson, and D. Snyder, "Distance in graphs," Czechoslovak Mathematical Journal, vol. 26, no. 2, pp. 283–296, 1976.
[14] B. Elspas and J. Turner, "Graphs with circulant adjacency matrices," Journal of Combinatorial Theory, vol. 9, no. 3, pp. 297–307, 1970.
[15] M. E. Muzychuk, M. H. Klin, and R. Pöschel, "The isomorphism problem for circulant graphs via schur ring theory." Codes and association schemes, vol. 56, pp. 241–264, 1999.
[16] B. Mans, "Optimal distributed algorithms in unlabeled tori and chordal rings," Journal of Parallel and Distributed Computing, vol. 46, no. 1, pp. 80–90, 1997.
[17] F. T. Leighton, Introduction to parallel algorithms and architectures: Arrays· trees· hypercubes. Elsevier, 2014.
[18] J.-C. Bermond, F. Comellas, and D. F. Hsu, "Distributed loop computer-networks: a survey," Journal of parallel and distributed computing, vol. 24, no. 1, pp. 2–10, 1995.
[19] S.-M. Tang, Y.-L. Wang, and C.-Y. Li, "Generalized recursive circulant graphs," IEEE Transactions on Parallel and Distributed Systems, vol. 23, no. 1, pp. 87–93, 2011.
[20] J.-S. Yang, J.-M. Chang, S.-M. Tang, and Y.-L. Wang, "On the independent spanning trees of recursive circulant graphs g (cdm, d) with d> 2," Theoretical Computer Science, vol. 410, no. 21-23, pp. 2001–2010, 2009.
[21] ——, "Constructing multiple independent spanning trees on recursive circulant graphs g (2m, 2)," International Journal of Foundations of Computer Science, vol. 21, no. 01, pp. 73–90, 2010.
[22] A. Zehavi and A. Itai, "Three tree-paths," Journal of Graph Theory, vol. 13, no. 2, pp. 175–188, 1989.

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

KAI-HSUN YAO received the B.S. degree from the Department of Mechanical and Electro-Mechanical Enginnering of National Sun Yat-sen University, Kaohsiung, Taiwan , in June 2014. He is studying assiduously for the M.S. degree in the Computer Science and Information Engineering Department of National Cheng Kung University, Tainan, Taiwan. His current research interests include design and analysis of algorithms and graph theory.

SUN-YUAN HSIEH received the PhD degree in computer science from National Taiwan University, Taipei, Taiwan, in June 1998. He then served the compulsory two-year military service. From August 2000 to January 2002, he was an assistant professor at the Department of Computer Science and Information Engineering, National Chi Nan University. In February 2002, he joined the Department of Computer Science and Information Engineering, National Cheng Kung University, and now he is a chair professor. His awards include the 2007 K. T. Lee Research Award, President's Citation Award (American Biographical Institute) in 2007, Engineering Professor Award of Chinese Institute of Engineers (Kaohsiung Branch) in 2008, National Science Council's Outstanding Research Award in 2009, IEEE Outstanding Technical Achievement Award (IEEE Tainan Section) in 2011, Outstanding Electronic Engineering Professor Award of Chinese Institute of Electrical Engineers in 2013, and Outstanding Engineering Professor Award of Chinese Institute of Engineers in 2014. He is Fellow of the British Computer Society (BCS) and Fellow of Institution of Engineering and Technology (IET).

Dr. Hsieh is also an experienced editor with editorial services to a number of journals, including serving as associate editors of IEEE ACCESS, IEEE Transactions on Reliability, Theoretical Computer Science (Elsevier), Discrete Applied Mathematics (Elsevier), Journal of Supercomputing (Springer), International Journal of Computer Mathematics (Taylor Francis Group), Parallel Processing Letters (World Scientific), Discrete Mathematics, Algorithms and Applications (World Scientific), Fundamental Informaticae (Polish Mathematical Society), and Journal of Interconnection Networks (World Scientific). In addition, he has served on organization committee and/or program committee of several dozens international conferences in computer science and computer engineering. His current research interests include design and analysis of algorithms, fault-tolerant computing, bioinformatics, parallel and distributed computing, and algorithmic graph theory.

• • •