The Constructive Algorithm of Vertex-disjoint Paths in the Generalized Hypercube under Restricted Connectivity

Guijuan Wang¹, Jianxi Fan¹, Yali Lv², Baolei Cheng³, Shuangxiang Kan¹

1 School of Computer Science and Technology, Soochow University, China
2 Institute of Information Technology, Henan University of Chinese Medicine, China
{guijuan_wang, lv_yali136}@126.com, {jxfan, chengbaolei}@suda.edu.cn, sxkan@foxmail.com

Abstract

The generalized hypercube is a classical interconnection network with excellent properties. It not only includes the hypercube network, the 3-ary n-cube network, and the complete networks, but also can be used to construct data center networks such as FBFLY, BCube, HyperX, SWCube, etc. Since the fact that all neighbors of one vertex becoming faulty at the same time is almost impossible, we assume that each vertex in this paper has at least one fault-free neighbor. We use $G(m_1, m_2, ..., m_r)$ to denote the r-dimensional generalized hypercube and $\kappa(G)$ to denote the 1-restricted connectivity of $G(m_1, m_2, ..., m_r)$. Then we design an algorithm to construct at least $\kappa(G)$ disjoint paths based on any two distinct vertices in $G(m_1, m_2, ..., m_r)$ under the 1-restricted connectivity. The maximum length of these disjoint paths is bounded by $r+2$.

Keywords: Generalized hypercube, Disjoint path, Restricted connectivity, Fault-tolerance

1 Introduction

A topology with excellent properties will improve the quality of an interconnection network. A good topology can make an interconnection network have low construction cost, low communication delay, high fault-tolerant ability, and so on. So far, researchers have proposed many excellent interconnection networks’ topologies such as the hypercube, the crossed cube, the twisted cube, the Möbius cube, etc. However, the representation of vertices in these networks is limited to binary only, which makes the topology not flexible. For letting the representation of vertices in networks no longer limited to binary and making the structure more general, Laxmi and Dharma proposed the generalized hypercube (GH) [1]. The generalized hypercube has excellent properties: it is easily to expand with recursive structure; it is edge symmetric and vertex symmetric; it has low communication delay, etc. It includes many interconnection networks such as: the hypercube, the completed network, the 3-ary n-cube and so on. Furthermore, it can be used to construct some data center networks [2-5]. Since the generality of the generalized hypercube, the study results about it can be applied into other networks. Therefore, there are many studies based on it [2-4, 6-8].

In this paper, we study the algorithm to construct vertex-disjoint paths in the generalized hypercube with 1-restricted connectivity. A topology of interconnection network can be modeled by a graph where vertex denotes the processor and edge denotes the communication link. That is, let $G=(V, E)$ denote an interconnection network, where $V$ and $E$ represent vertex set and edge set, respectively. We use $\kappa(G)$ to denote the connectivity of a graph $G$, which is the minimum number of vertices in set $S \subset V$ and the graph $G$ is disconnected when deleting $S$. We can estimate the communication capability of vertices by connectivity and we can also use it to measure the fault-tolerant ability of one network. However, many works assume that the neighbors of one vertex can become faulty at the same time when estimating the fault-tolerant ability based on the connectivity, which has quite low probability. Therefore, in order to more accurately measure the communication and fault-tolerant ability of one network based on connectivity, a lot of conditions are added into the connectivity. Esfahanian and Hakimi introduced the concept of restricted connectivity [9-10]. Let $\kappa^\delta(G)$ be the g-restricted connectivity of $G$, which is the minimum number of vertices in set $F \subset V$, whose deletion disconnects $G$ and each vertex has at least $g$ fault-free neighbors in each disconnected component. So far, there are many studies based on the restricted connectivity. Chen et al. studied the restricted vertex connectivity and the restricted edge connectivity of three families of interconnection networks [11]. Hsieh et al. studied the $\{2, 3\}$-restricted connectivity of...
locally twisted cubes [12]. Wang et al. proved that the \( h \)-restricted connectivity of the data center network DCell is almost as \((h+1)\) times as traditional connectivity [13-14]. Balbuena and Marcote studied the \( p \)-restricted edge-connectivity of Kneser graphs [15]. Then it has received many attentions from outstanding researchers [16-22]. In this paper, we assume that each vertex has at least one fault-free neighbor which can better reflect the actual communication of a network.

Vertex-disjoint paths are those that do not share any common vertex except for end vertices. Disjoint paths are fundamental and essential for parallel, distributed computing, fault-tolerance, and load balancing of a network [23]. For transmitting data in the network stably and safely, more and more works are based on vertex-disjoint paths. Lai studied the optimal construction of all shortest vertex-disjoint paths in the hypercube with applications [24]. Furthermore, the maximal length of paths is also minimized in the worst case. Cheng et al. proposed an \( O(N\log N) \) recursive algorithm to construct \( n \) independent spanning trees in \( \text{Möbius} \) cube, and further they constructed \( n \) vertex-disjoint paths based on the \( n \) independent spanning trees [25]. Cheng et al. proved that there exist two vertex-disjoint paths in the balanced hypercube, and then they studied the hamiltonian laceability of the balanced hypercube based on vertex-disjoint paths [26]. Inoue proved that network reliability and the criticality of links are greatly dependent on path disjointness [27]. However, these works do not consider the restricted connectivity when construct vertex-disjoint paths.

In this paper, we propose algorithms to construct vertex-disjoint paths based on any two distinct vertices under \( 1 \)-restricted connectivity. We use \( G(m_1,m_2,\ldots,m_r) \) to denote the \( r \)-dimensional generalized hypercube and \( \kappa_1(G) \) to be the \( 1 \)-restricted connectivity. In this paper, we proposed an algorithm to construct at least \( \kappa_1(G) \) disjoint paths based on any two distinct vertices in the generalized hypercube under \( 1 \)-restricted connectivity in \( O(nr) \) time, where the maximum length of these disjoint paths is bounded by \( r+2 \). The rest of this paper is organized as follows. In Section 2, we give some basic definitions and notations used in this paper and some properties of the generalized hypercube. Then, we design the construction algorithms to construct vertex-disjoint paths in Section 3. In Section 4, we do simulations to analyze performances of the proposed algorithm. Finally, we provide conclusions of the paper in Section 5.

2 Preliminaries

In this section, we first introduce some definitions and notations used in this paper and introduce the definition and properties of the generalized hypercube.

2.1 Definitions and Notations

Given an undirected simple graph \( G=(V(G),E(G)) \), where \( V(G) \) and \( E(G) \) represent vertex set and edge set, respectively. Let \((u,v)\) be an edge with end vertices \( u \) and \( v \). If \((u,v)\in E(G)\), we call \( u \) and \( v \) are neighbors for each other. Let \( P(u',u'')= (u' \rightarrow u'' \rightarrow \ldots \rightarrow u''') \) be a path from \( u' \) to \( u'' \) in which any two consecutive vertices are adjacent. Let \( P(u',u')= (u' \rightarrow u'' \rightarrow \ldots \rightarrow u'') \). Then we call path \( P(u',u'') \) to be the sub-path of \( P(u',u'') \). Furthermore, we write \( P(u',u'')= (u' \rightarrow \ldots u'' \rightarrow u'') \rightarrow P(u',u'') \rightarrow u'' \). We use \( F \) to represent the faulty vertices set of \( G \). If a vertex \( u \) is faulty vertex; otherwise we call it fault-free. If each vertex has one fault-free neighbor in graph \( G\cdot F \), we call the connectivity of \( G \) under this condition as the \( 1 \)-restricted connectivity, denoted by \( \kappa_1(G) \).

The generalized hypercube is a general class of hypercube structures which is designed to be used in the parallel and distributed environments [1]. Then we give the definition of an \( r \)-dimensional generalized hypercube as follows:

**Definition 1.** For any integer \( r \geq 1 \), an \( r \)-dimensional generalized hypercube, denoted by \( G(m_1,m_2,\ldots,m_r) \), has \( \prod_{i=1}^{r} m_i \) vertices, where \( m_i \) is the number of vertices in each dimension. Each vertex \( u \) in \( G(m_1,m_2,\ldots,m_r) \) can be denoted by an \( r \)-digit identifier \( u_{i_1}u_{i_2}\ldots u_{i_r} \), where \( 0 \leq u_i \leq m_i-1 \) with \( 1 \leq i \leq r \). Two vertices in \( G(m_1,m_2,\ldots,m_r) \) are adjacent if and only if their identifiers differ at exactly one position.

Figure 1 demonstrates the structure of \( G(3,4) \) and \( G(3,3,4) \). We know that \( G(3,3,4) \) is constructed by 3 \( G(3,4) \) s. Therefore, one \( G(m_1,m_2,\ldots,m_r) \) is made up of \( m_i \) \( G(m_{i-1},m_{i-2},\ldots,m_1) \).

![Figure 1](image)

(a) (b)

Figure 1. The structure of \( G(3,4) \) and the structure of \( G(3,3,4) \).

According to [1], we have the following theorems.
2.2 Properties of $G(m_1, m_2, \ldots, m_r)$ Under 1-restricted Connectivity

In this paper, we assume that each vertex in $G(m_1, m_2, \ldots, m_r)$ has at least one fault-free neighbor. For simplicity, let $G$ represent $G(m_1, m_2, \ldots, m_r)$ in the following section and these two symbols can be used alternately. Given two arbitrary vertices $u = u_1 u_2 \ldots u_r$ and $v = v_1 v_2 \ldots v_r$ in $G(m_1, m_2, \ldots, m_r)$. Let $x = u_1 u_2 \ldots u_{i-1} u_{i+1} \ldots u_r$ be a $u$'s fault-free neighbor, $u_i \neq g$; and $y = v_1 v_2 \ldots v_{i-1} v_{i+1} \ldots v_r$ be a $v$'s fault-free neighbor, $v_i \neq g$. We use an array $L_{uv} = [l_0, l_1, \ldots, l_r]$ to indicate positions at which $u$ and $v$ have different value.

For example, when $u = 000000$ and $v = 003102$, then $L_{uv} = [4, 3, 1]$ and $\alpha = 3$, since at positions 4, 3, and 1, $u$ and $v$ have different bits. If $u$ and $v$ are adjacent, then they have exactly one different bit. Furthermore, we use *hamming distance* to represent the distance between $u$ and $v$, denoted by $h(u, v)$, which is defined as the cardinality of $\{ i | u_i \neq v_i \}$ [28], i.e., $h(u, v) = L_{uv} = \alpha$. In this paper, we consider paths between $u$ and $v$ whose distance is at least 2, i.e., $\alpha \geq 2$ since we already proved that we can design an algorithm to construct at least $\kappa'(G)$ disjoint paths when $\alpha=1$ and this result has been accepted by hpcc2019 conference.

According to [29], we have the following theorem:

**Theorem 3.** The 1-restricted connectivity of $G(m_1, m_2, \ldots, m_r)$ is $\kappa'(G) = 2\kappa(G) - n$, where $n = \max \{ m_i | 1 \leq i \leq r \}$.

3 Disjoint Paths

In this section, we design algorithms to construct vertex-disjoint paths in $G$ under 1-restricted connectivity. Given any two vertices $u$ and $v$, and their neighbors $x$ and $y$, we can use four kinds of methods to construct disjoint paths which end vertices are $u$ and $v$. Since we assume that each vertex in $G$ has at least fault-free neighbor, therefore paths constructed in this paper may contain the neighbors of end vertices. In there, we redefine the definition of vertex-disjoint paths: paths are vertex-disjoint if they have no common vertices other than vertex in set $\{u, x, y, v\}$.

The first way is to construct paths which do not pass through vertex $x$ as shown in Figure 2(a). The second way is to construct paths which must pass through $x$ as shown in Figure 2(b). The third way is to construct paths which do not pass through $x$ but must pass through $y$ as shown in Figure 2(c). The fourth way is to construct paths which pass through vertices $x$ and $y$ at the same time, as shown in Figure 2(d).

**Figure 2.** Four kinds of disjoint paths.

3.1 The Base Paths

Firstly, we introduce how to construct path between two vertices by Algorithm 1. The process of path construction is just a process of bit-changing. For example, let $u = 000000$, $v = 003102$, then one of paths between $u$ and $v$ is $0000 \rightarrow 0003 \rightarrow 0033 \rightarrow 0433$. Let $P$ be a path from $u$ to $v$ and the number of different bits between two vertices satisfies $2 \leq \alpha \leq r$. So in $P$ there will be at least $\alpha$ bit-changes, meaning the length of $P$ is at least $\alpha$. Our target is to construct vertex-disjoint paths between $u$ and $v$, and our final algorithm constructs each path by splicing a few (up to 3) sub-paths.

As just mentioned, going from a vertex $u$ to the next vertex is equivalent to changing one bit of $u$. So by selecting the bits to change in certain order, we are actually selecting a particular path. In Algorithm 1, when specifying a sub-path, we give the bit-location to start the change (denoted by $s$) and the bit-location to terminate the change (denoted by $t$). For example, let $u = 000000$, and $v = 032433$. Then the different bit position array $L_{uv}$ is $L_{uv} = [5, 4, 3, 2, 1]$. Let $s = 1$, $t = 3$, which means that the path starts by first changing the bit at location $l_1$, then $l_2$, and ends at $l_3$. Thus, the corresponding sub-path is $u = 000000 \rightarrow 00003 \rightarrow 00033 \rightarrow 00433$. (Note: it has not reached $v$ yet since it is a sub-path.)

Algorithm 1 (PA) as below is the pseudocode to describe the procedure to construct up to three sub-paths from vertex $z_0$ to vertex $z_1$. Since the path constructed by Algorithm 1 may be just a sub-path of $P_{uv}$, vertex $z_0$ is not necessarily $u$ and vertex $z_1$ is not necessarily $v$. PA’s inputs $s_1$, $s_2$ and $s_3$ are the
three start bit-locations, while \( t_1, t_2 \) and \( t_3 \) are the three terminal bit-locations. Lines 2--5 calculate the length for each of three sub-paths. Then lines 6--8 describe how to change bits in the 1st sub-path. Similarly, lines 9--11 describe how to change bits in the 2nd sub-path, and lines 12--14 describe how to change bits in the 3rd sub-path. Finally, the fifth line outputs the constructed paths.

**Algorithm 1:** \( PA(z_0, v; L_{uv}, s_1, t_1, s_2, t_2, s_3, t_3) \)

Input: source vertex \( z_0 \), terminal vertex \( v \), the array \( L_{uv} \), and indexes \( s_1, t_1, s_2, t_2, s_3, t_3 \).

Output: a path from \( z_0 \) to \( v \), \( t = m_1 + m_2 + m_3 \).

begin
for \( i = 1 \) to 3 do
if \( s_i \neq 0 \) then \( m_i = \mid t_i - s_i \mid + 1 \);
else \( m_i = 0 \);
end for
if \( s_1 \neq 0 \) then
\[ z_i = (z_{i-1})_{l_{i-1}+1}^{y_{i-1}+1} \] for \( 1 \leq i \leq m_1 \);
end if
if \( s_2 \neq 0 \) then
\[ z_i = (z_{i-1})_{l_{i-2}+1}^{y_{i-2}+1} \] for \( m_1 + 1 \leq i \leq m_1 + m_2 \);
end if
if \( s_3 \neq 0 \) then
\[ z_i = (z_{i-1})_{l_{i-3}+1}^{y_{i-3}+1} \]
\[ m_1 + m_2 + 1 \leq i \leq m_1 + m_2 + m_3 \];
end if
end return \( (z_0, z_1, ..., z_m, s_{23}, t_1, t_2, t_3) \);
end

Since bit-changing is a basic operation in Algorithm 1, we use \( u_i \) to denote the \( i \)-th bit of vertex \( u \) and we use \( u_j \) to denote the vertex obtained by changing \( u_i \)'s \( j \)-th bit to \( j \). For example, if \( u = 010203 \), then \( u_1 = 3 \), \( u_4 = 010204 \), \( u_5 = 2 \), and \( u_8 = 010403 \). In Algorithm 1, when constructing the first sub-path, vertex \( z_i = (z_{i-1})_{l_{i-1}+1}^{y_{i-1}+1} \) for \( 1 \leq i \leq m_1 \). For example, let \( u = 000000 \), \( v = 113112 \), \( s_1 = 2, t_1 = 4 \), and \( z_0 = u \), then \( L_{uv} = [6, 5, 4, 3, 2, 1] \) and \( m_1 = 3 \). The \( z_1 = (z_0)_{l_1}^{y_1} \), where \( l_{2}^{y_1} = l_2 = 2 \) and \( v_{l_2} = v_2 = 1 \); \( z_1 = (z_0)_{s_1}^{t_1} = (000000)_{t_1} = 000110 \). Similarly, \( z_2 = (z_1)_{l_2}^{y_2} = (z_0)_{l_2}^{y_2} = (000110)_{y_2} = 0001110 \). The first sub-path is \( 000000 \rightarrow 000110 \rightarrow 0001110 \rightarrow 003110 \). Then the changing of vertices in the second sub-path and the third sub-path are similarly.

Then we use an example to illustrate the construction path process of our algorithm. For example, let \( u = 000000, v = 113112 \), and \( z_0 = u \), then \( L_{uv} = [6, 5, 4, 3, 2, 1] \). If the whole path contains just one sub-path, then path \( PA(z_0, v; L_{uv}, 1, 6, 0, 0, 0, 0) = 000000 \rightarrow 000002 \rightarrow 000012 \rightarrow 000112 \rightarrow 003112 \rightarrow 013112 \rightarrow 113112 \).

The following is a path composed of two sub-paths: \( PA(z_0, v, L_{uv}, 4, 6, 1, 3, 0, 0) = 000000 \rightarrow 003000 \rightarrow 013000 \rightarrow 113000 \) and \( 113002 \rightarrow 113012 \rightarrow 113112 \). And a path composed of three sub-paths: \( PA(z_0, v, L_{uv}, 5, 6, 3, 4, 1, 2) = 000000 \rightarrow 010000 \rightarrow 110000 \rightarrow 110100 \rightarrow 111000 \rightarrow 111010 \rightarrow 11102 \rightarrow 111012 \rightarrow 113112 \).

From the structural process above, we can see that the maximum length of paths constructed by Algorithm 1 is \( h(u, v) \).

Then according to the construction process of Algorithm 1, we can get the following lemma.

**Lemma 1.** Vertices in each path constructed by Algorithm 1 are different.

**Proof.** In Algorithm 1, we let that \( \{ s_1, s_1 + 1, ..., t_1 \} \cap \{ s_2, s_2 + 1, ..., t_2 \} \cap \{ s_3, s_3 + 1, ..., t_3 \} \neq \emptyset \), which denotes that the location of bit-changing is different for all vertices in each path. Therefore, vertices \( z_0, z_1, ..., z_n \) are different.

The lemma holds.

### 3.2 The First Method to Construct Paths

In this section, we design Algorithm 2 to construct vertex-disjoint paths from \( u \) to \( v \), where all paths circumventing a particular vertex \( x \) which is a neighbor of \( u \). We use \( P_l \) to denote the path set obtained by Algorithm 2. The input of \( u, v \) are two end vertices of paths obtained by Algorithm 2, \( k \) is the location of the bit at which \( u \) and \( v \) are different, \( r \) is the dimension of the generalized hypercube, \( G \) represents \( G(m_1, m_2, ..., m_l) \), and \( L_{uv} \) is the position array of different bits between \( u \) and \( v \). The meaning of these parameters in following algorithms is the same.

In Algorithm 2, the line 2 assigns values to each parameter. Then lines 3--15 construct paths from \( u \) to \( v \). The line 16--19 determines whether \( k \) belongs to \( L_{uv} \). Finally, the line 20 outputs the path set \( P_l \) and deletes the path that passes through vertex \( x \).

Note that we can start a bit-changing process from any bit. For \( 1 \leq i \leq r \), let the starting bit be \( i \), and let \( P_i \) represent the set of all paths with starting bit \( i \).

**Algorithm 2:** \( BP2-1(u, v, G, k, L_{uv}, 1, r) \)

Input: vertices \( u \) and \( v \), the graph \( G \) and the array \( L_{uv} \), and indexes \( k, l, r \).

Output: disjoint paths from \( u \) to \( v \), which do not pass through \( x \).

begin
\( P_l \leftarrow \emptyset \), \( \alpha = |L_{uv}| \), \( s_1 = s_2 = t_2 = s_3 = t_3 = 0 \), \( t_1 = \alpha \).
for $i=1$ to $r$ do  
for $j=0$ to $m_i-1$ do  
if $u_j \neq j$ then  
if $i \notin L_{uv}$ then  
$P_i = P_i \cup \{u, PA(u_{i}^{w} \varepsilon, v, L_{uv}, 1, \alpha-1, 0, 0, 0, 0), v\}$;  
else  
let $s_i$ be the index such that $l_{s_i} = i$;  
$s_1 = 1, t_2 = s_1 - 1$;  
$P = P \cup u, PA(u_{i}^{w} \varepsilon, v, L_{uv}, s_1 + 1, t_1, t_2, t_3, t_4, t_5), v)\};$  
end if  
end if  
end for  
end for  
if $k \in L_{uv}$ then  
let $s_i$ be the index such that $l_{s_i} = k$;  
$s_2 = 1, t_3 = s_2 - 1$;  
end if  
return $P_i = \{(u, PA(u_{i}^{w} \varepsilon, v, L_{uv}, s_1 + 1, t_1, t_2, t_3, t_4), v)\};$  
end

Algorithm 2 will call Algorithm 1 (PA). If we start a bit-changing process from $i$-th bit, then the range of the $i$-th bit of $u_{i}^{w} \varepsilon$ is $[0, m_i - 1]$. Therefore, the number of paths in $P_i$ is $m_i - 1$. The number of paths in $P_i$ is $\sum_{i=1}^{r} (m_i - 1) = \kappa(G) - 1$. Then deleting the path that passes through $x$, and the number of paths constructed by Algorithm 2 is $\kappa(G) - 1$.

Then we will see how the algorithm works by going over an example. We set the generalized hypercube is $G(4, 4, 4, 4, 4)$. Let $u = 00000$, $v = 00111$, $x = 00010$. Then $L_{uv}=[3,2,1]$. We have $k=2$, at which $u$ and $x$ differ, and $k \in L_{uv}$. According to BP2-1, we can construct paths with the value of $i$ from 1 to 5. We know that when $i \in L_{uv}$ the values of $i$ are 1, 2, 3 and when $i \notin L_{uv}$ the values of $i$ are 4, 5. Then we can get the path sets as follows:

$P_1 = \{(00000 \to 00000 \to 00000 \to 00000 \to 00000 \to 00111) \}$

$|0 \leq j \leq m_1 - 1, j \neq u_1\} = \{(u, PA(u_{i}^{w} \varepsilon, v, L_{uv}, 2, 3, 0, 0, 0, 0, 0), v) | 0 \leq j \leq m_i - 1, j \neq u_1\}$.

$P_2 = \{(00000 \to 00000 \to 00000 \to 00000 \to 00000 \to 00111)\}$

$|0 \leq j \leq m_2 - 1, j \neq u_2\} = \{(u, PA(u_{i}^{w} \varepsilon, v, L_{uv}, 3, 3, 1, 1, 1, 0, 0, 0), v) | 0 \leq j \leq m_2 - 1, j \neq u_2\}$.

$P_3 = \{(00000 \to 00000 \to 00000 \to 00000 \to 00000 \to 00111)\}$

$|0 \leq j \leq m_3 - 1, j \neq u_3\} = \{(u, PA(u_{i}^{w} \varepsilon, v, L_{uv}, 4, 4, 4, 4, 4, 4, 4, 4), v) | 0 \leq j \leq m_3 - 1, j \neq u_3\}$.

$P_4 = \{(00000 \to 00000 \to 00000 \to 00000 \to 00000 \to 00111)\}$

$|0 \leq j \leq m_4 - 1, j \neq u_4\} = \{(u, PA(u_{i}^{w} \varepsilon, v, L_{uv}, 5, 5, 5, 5, 5, 5, 5, 5), v) | 0 \leq j \leq m_4 - 1, j \neq u_4\}$.

$P_5 = \{(00000 \to 00000 \to 00000 \to 00000 \to 00000 \to 00111)\}$

$|0 \leq j \leq m_5 - 1, j \neq u_5\} = \{(u, PA(u_{i}^{w} \varepsilon, v, L_{uv}, 6, 6, 6, 6, 6, 6, 6, 6), v) | 0 \leq j \leq m_5 - 1, j \neq u_5\}$.

We know $k=2$, then we need to delete the path that passes through $x$, $p = (00000 \to 00010 \to 00110 \to 00111)$. Then $P_2^r = P_2^r - p$. Therefore, the path set $P_i = P_1 \cup P_2^r \cup P_3^r \cup P_4^r \cup P_5^r$.

We will prove paths constructed by Algorithm 2 are disjoint. Then we have the following lemma.

**Lemma 2.** For $1 \leq i \leq r$, Algorithm 2 constructs at least $\kappa(G) - 1$ disjoint paths based on $u$ and $v$ which do not pass through vertex $x$.

**Proof.** We use $N_i$ to represent the number of paths in $P_i$. Then $N_i = \sum_{i=1}^{r} (m_i - 1) = \kappa(G) - 1$. We let $Q_1$ and $Q_2$ be two different paths, where $Q_1, Q_2 \in P_i$. We will prove that paths $Q_1$ and $Q_2$ are disjoint. There are two cases.

Case 1: For $1 \leq i \leq r$, $Q_1, Q_2 \in P_i$. Since $Q_1$ and $Q_2$ are two different paths, the values of $i$-th bit are different in $Q_1$ and $Q_2$. Therefore, for any two vertices $v_1 \in V(Q_1)$ and $v_2 \in V(Q_2)$, we have $v_1 \neq v_2$. Thus, paths $Q_1$ and $Q_2$ are disjoint.

Case 2: For $1 \leq i \neq w \leq r$, $Q_1 \in P_i$ and $Q_2 \in P_w$. For any two different vertices $v_1$ and $v_2$ such that $v_1 \in V(Q_i)$ and $v_2 \in V(Q_w)$, we have three subcases as follows.

Case 2.1: $i \notin L_{uv}$ and $w \notin L_{uv}$. Vertices in paths $Q_1$ and $Q_2$ are different in $i$-th and $w$-th bits. Namely, $v_1 \neq v_2$ and $v_{1i} \neq v_{2i}$. Therefore, paths $Q_1$ and $Q_2$ are disjoint.

Case 2.2: $i \in L_{uv}$, $w \notin L_{uv}$ or $i \notin L_{uv}$, $w \in L_{uv}$. Without loss of generality, we let $i \in L_{uv}$, $w \notin L_{uv}$, then vertices in $Q_1$ and $Q_2$ are different in $i$-th bit. Namely, $v_{1i} \neq v_{2i}$. Therefore, paths $Q_1$ and $Q_2$ are disjoint.

Case 2.3: $i, w \in L_{uv}$. Since $i \neq w$, according to Algorithm 2, the first position of variable bit and the last position of variable bit in two paths are different. Then, since the ordering of variable bit in two paths are the same, there is no vertex that is the same in two paths. Let $l_{pi} = i$ and $l_{pw} = w$. It is obvious that $v_{1i} \neq v_{2i}, v_{1j} \neq v_{2j}$, $v_{1j} \neq v_{2j}$, $v_{1j} \neq v_{2j}$. Therefore, paths $Q_1$ and $Q_2$ are disjoint.

The lemma holds.

### 3.3 The Second Method to Construct Paths

We design Algorithm 3 to construct vertex-disjoint paths from $u$ to $v$ in which all paths pass through vertex $x$. In Algorithm 3, lines 1–2 assign values to each parameter. Then lines 3–41 construct paths from $x$ to $v$. Finally, the line 42 outputs the path set $P_i$. In this section, we use $P_i$ to denote the path set obtained by Algorithm 3.
Algorithm 3: $BP2 - 2(u, v, x, G, k, L_w, 1, r)$

Input: vertices $u$ and $v$, the graph $G$ and the array $L_w$ and indexes $k, 1, r$.

Output: disjoint paths from $u$ to $v$ which all pass through $x$.

begin

$P_2 \leftarrow \emptyset$, $\alpha = |L_w|$, $s_1 = s_2 = a_1 = a_2 = 0$, $n_0 = x_i^j$, $n_k = (n_k)^{i^{01}}$, $n_k = (n_k)^{i^{02}}$;

for $i = 1$ to $r$ do

for $j = 0$ to $m_i - 1$ do

if $k \in L_w$ then

let $s_1$ be the index such that $L_i = k$;

end if

if $i \in L_w$ then

let $s_2$ be the index such that $L_i = i$;

end if

if $i \neq k$ & & $u \neq j$ then

if $|L_w| = 3$ & & $k \in L_w$ & & $i = \max \{L_w \setminus \{k\}\}$ then

$L_i = \{L_w \setminus \{k, i\}\}$;

$P_2 = P_2 \cup \{u, x, n_0, P_A((n_k)^{i^{01}}), L_w, m, m, 0, 0, 0, 0, 0\}$;

else

if $s_1 == 0$ then

$a_1 = s_1$, $a_2 = a_0 = 0$, $b_1 = a$, $b_2 = b_0 = 0$;

else

$a_1 = s_1$, $a_2 = 1$, $a_0 = 0$, $b_1 = a$, $b_2 = b_0 = 0$;

end if

else

if $s_2 == 0$ then

$a_1 = s_1$, $a_2 = 1$, $a_0 = 0$, $b_1 = a$, $b_2 = s_2 - 1$, $b_0 = 0$;

end if

else

if $s_1 > s_2$ then

$a_1 = s_1$, $a_2 = 1$, $a_0 = 0$, $b_1 = a$, $b_2 = s_2 - 1$, $b_0 = s_1 - 1$;

else

$a_1 = s_1$, $a_2 = 1$, $a_0 = 0$, $b_1 = a$, $b_2 = s_2 - 1$, $b_0 = s_1 - 1$;

end if

end if

end if

end for

for $i$ do

end if

end for

end

end

For $1 \leq i \leq r$ and $i \neq k$, we let the starting bit be $i$, and let $P_i$ represent the set of all paths with starting bit $i$. Algorithm 3 will call Algorithm 1 (PA). If we start a bit-changing process from bit $i$, the range of the value of $u_i$ is $[0, m_i - 1 \setminus \{u_i\}]$. Therefore, the number of paths in $P_i$ is $m_i - 1$. The number of paths in $P_i$ is $\sum_{i = 1}^{r} (m_i - 1) = \kappa(G) - m_r + 1$.

Then we will see how the algorithm works by going over an example. We set the generalized hypercube is $G(4, 4, 4, 4, 4)$. Let $u = 00000$, $v = 00111$, $x = 00010$. Then $L_w = [3, 2, 1]$. We have $k = 2$, at which $u$ and $x$ differ, and $k \in L_w$. According to the algorithm, we can construct paths with the value of $i$ from 1 to 5 and $i \neq 2$. We know that $i \in L_w$, the values of $i$ are 1, 2, 3 and when $i \in L_w$, the values of $i$ are 4, 5. Then we can get the path sets as follows:

$P_1 = \{(00000 \rightarrow 00010 \rightarrow 00111 \rightarrow 00111 \rightarrow 00111) \}$

$|0 \leq j \leq m_i - 1, j \neq u_i\} = \{(u, x, x_i^j, (x_i^j)^{01}, (x_i^j)^{02})^j, PA(((x_i^j)^{01})^j(u, x, x_i^j, n_0, P_A((n_k)^{i^{01}}), L_w, m, m, 0, 0, 0, 0, 0), v) |0 \leq j \leq m_i - 1, j \neq u_i\}$.

$P_2 = \{(00000 \rightarrow 00010 \rightarrow 00111 \rightarrow 00111 \rightarrow 00111) \} \}$

$|0 \leq j \leq m_i - 1, j \neq u_i\} = \{(u, x, x_i^j, (x_i^j)^{01}, (x_i^j)^{02})^j, PA(((x_i^j)^{01})^j(u, x, x_i^j, n_0, P_A((n_k)^{i^{01}}), L_w, m, m, 0, 0, 0, 0, 0), v) |0 \leq j \leq m_i - 1, j \neq u_i\}$.

Therefore, the path set $P_2 = P_2 \cup P_2 \cup P_2 \cup P_2$.

We will prove paths constructed by Algorithm 3 are disjoint. Then we have the following lemma.

Lemma 3. Algorithm 3 can construct $\kappa(G) - m_k + 1$ disjoint paths based on $u$ and $v$ in which each path passes through $x$.

Proof. We use $N_2$ to denote the number of paths in $P_z$. Then we set $P_z$ to be the $i$-th path set of $P_z$, where $1 \leq i \leq r$ and $i \neq k$. We let $Q_1$ and $Q_2$ be two different paths, where $Q_1, Q_2 \in P_z$. We will prove that paths $Q_1$ and $Q_2$ are disjoint. There are two cases.

Case 1: For $1 \leq i \leq r$ and $i \neq k$, $Q_1, Q_2 \in P_z$. The values of $j$ in $i$-th bit are different in $Q_1$ and $Q_2$. Therefore, vertices in $Q_1$ and $Q_2$ are different in $i$-th bit. Thus, paths $Q_1$ and $Q_2$ are disjoint.

Case 2: For $1 \leq i \neq r$ and $i \neq k$, $w \neq k$, $Q_1 \in P_z$ and $Q_2 \in P_z$ for any two different vertices $v_1$ and $v_2$.
such that \( v_1 \in V(Q_1) \) and \( v_2 \in V(Q_2) \), we have three subcases as follows:

Case 2.1: \( i \notin L_{uv} \) and \( w \notin L_{uv} \). Vertices in paths \( Q_1 \) and \( Q_2 \) are different in \( i \)-th and \( w \)-th bits. Namely, \( v_1, \neq v_2 \), and \( v_1, w, \neq v_2, w \). Therefore, paths \( Q_1 \) and \( Q_2 \) are disjoint.

Case 2.2: \( i \in L_{uv}, w \notin L_{uv} \) or \( i \notin L_{uv} \). Without loss of generality, we let \( i \in L_{uv}, w \notin L_{uv} \) then vertices in \( Q_1 \) and \( Q_2 \) are different in \( i \)-th bit. Namely, \( v_1, \neq v_2 \). Therefore, paths \( Q_1 \) and \( Q_2 \) are disjoint.

Case 2.3: \( i, w \in L_{uv} \). When \( k \notin L_{uv} \), the proof is similar to that of case 2.3 in Lemma 2.

The lemma holds.

3.4 The Third Method to Construct Paths

In this section, we design Algorithm 4 to construct vertex-disjoint paths from \( u \) to \( v \) in which all paths circumventing vertex \( x \) and pass through \( v \)'s neighbor \( y \). We use \( P_1 \) to denote the path set obtained by Algorithm 4.

**Algorithm 4: BP2−3(u,v,y;G,n,Luv,1,r)**

*Input*: vertices \( u, v \) and \( y \), the graph \( G \) and the array \( L_{uw} \), and indexes \( n, l, r \).

*Output*: disjoint paths from \( u \) to \( v \) which all pass through \( x \).

```
begin

\( P_1 \leftarrow \emptyset \); \( a = |L_{uv}| \); \( s_1 = s_2 = 0 \);

for \( i = 1 \) to \( r \) do

for \( j = 0 \) to \( m_i - 1 \) do

if \( n \in L_{uv} \) then

let \( s_i \) be the index such that \( l_i = n \);

end if

if \( i \in L_{uv} \) then

let \( s_i \) be the index such that \( l_i = i \);

end if

if \( i \neq n \) \&\& \( u_i \neq j \) then

if \( s_1 = 0 \) \&\& \( s_2 = 0 \) then

\( a_i = a_1 = a_2 = b_1 = b_2 = 0 \);

end if

if \( s_1 = 0 \) \&\& \( s_2 > 0 \) then

\( a_i = a_1 = a_2 = 0, b_1 = a, b_2 = s_2 - 1, b_3 = 0 \);

end if

if \( s_1 > 0 \) \&\& \( s_2 = 0 \) then

\( a_i = a_1 = a_2 = 0, a_3 = 0, b_1 = a, b_2 = s_2 - 1, b_3 = 0 \);

end if

if \( s_1 = s_2 \) then

\( a_i = a_1 = a_2 = s_1 + 1, b_1 = a, b_2 = s_2 - 1, b_3 = s_1 - 1 \);

else

\( a_i = a_1 = a_2 = s_1 + 1, a_3 = 0, b_1 = s_2 - 1, b_2 = a, b_3 = s_1 - 1 \);

end if

end if

\( P_1 = P_1 \cup \{u_i, u_i, PA((u_i)_u, v, L_{uw}, 3, 3, 1, 1, 0, 0, y, v) | 0 \leq j \leq m_i - 1, j \neq u_i \} \).

end if

end for

end for

end for

end for
```

In Algorithm 4, the line 1 assigns values to each parameter. Then lines 3--33 construct paths from \( u \) to \( v \) which circumvents \( x \) and passes through \( y \). Finally, line 34 outputs the path set \( P_1 \). For \( 1 \leq i \leq r \) and \( i \neq n \), let this starting bit be \( i \), and let \( P_i \) represent the set of all paths starting with bit \( i \). Algorithm 4 will call Algorithm 1 (PA). If we start a bit-changing process from bit \( i \), then the range of the value of \( u_i \) is \([0, m_i - 1] \setminus \{u_i\}\).

Therefore, the number of paths in \( P_i \) is \( m_i - 1 \). The number of paths in \( P_i \) is \( \sum_{i=0}^{m_i} (m_i - 1) = k(G) \). Then deleting the path that passes through \( x \), the number of paths constructed by Algorithm 4 is \( k(G) - 1 \).

Then we will see how the algorithm works by going over an example. We set the generalized hypercube is \( G(4, 4, 4, 4, 4) \). Let \( u = 00000, v = 00111, x = 00001 \), and \( y = 00121 \). Then \( L_{uv} = [3, 2, 1] \). We have \( k = 1 \) and \( n = 2 \) at which \( y \) and \( v \) differ, and \( k, n \in L_{uv} \). According to the algorithm, we can construct paths with the value of \( i \) from 1 to 5. We know that when \( i \in L_{uv} \) the values of \( i \) are 1, 2, 3 and when \( i \notin L_{uv} \) the values of \( i \) are 4, 5. Then we can get the path sets as follows:

\( P_1 = \{(00000 \rightarrow 00000 \rightarrow 00121 \rightarrow 00111) | 0 \leq j \leq m_i - 1, j \neq u_i \} \).

\( P_2 = \{(00000 \rightarrow 00000 \rightarrow 00121 \rightarrow 00111) | 0 \leq j \leq m_i - 1, j \neq u_i \} \).

\( P_3 = \{(00000 \rightarrow 00000 \rightarrow 00121 \rightarrow 00111) | 0 \leq j \leq m_i - 1, j \neq u_i \} \).

\( P_4 = \{(00000 \rightarrow 00000 \rightarrow 00121 \rightarrow 00111) | 0 \leq j \leq m_i - 1, j \neq u_i \} \).

\( P_5 = \{(00000 \rightarrow 00000 \rightarrow 00121 \rightarrow 00111) | 0 \leq j \leq m_i - 1, j \neq u_i \} \).

We know \( k = 2 \), then we need delete the path that passes through \( x \), \( p = \{00000 \rightarrow 00010 \rightarrow 00110 \rightarrow 00111\} \). Then \( P_i = P_i - p \). Therefore, the path set \( P = P_1 \cup P_2 \cup P_3 \cup P_4 \cup P_5 \).

We will prove paths constructed by Algorithm 4 are
disjoint. Then we have the following lemma.

**Lemma 4.** Algorithm 4 can construct \( \kappa(G) - 1 \) disjoint paths based on \( u \) and \( v \) in which each path passes through \( x \) and passes through \( y \).

**Proof.** We use \( N_i \) to denote the number of paths in set \( P_i \). Thus, \( N_i = (\sum_{j=1}^{r} (m_j - 1)) - 1 = \kappa(G) - 1 \). Then we set \( P_i' \) to be the \( i \)-th path set of \( P_i \), where \( 1 \leq i \leq r \). We let \( Q_1 \) and \( Q_2 \) be two different paths, where \( Q_1, Q_2 \in P_i \). We will prove that paths \( Q_1 \) and \( Q_2 \) are disjoint. The proof is similar to Lemma 3.

The lemma holds.

### 3.5 The Fourth Method to Construct Paths

Finally, we design Algorithm 5 to construct vertex-disjoint paths from \( u \) to \( v \) in which each path passes through \( x \) and \( y \). In Algorithm 5, the lines 1--2 assign values to each parameter and define the values of four vertices respectively. Then lines 3--34 construct paths from \( u \) to \( x \) then to \( y \) to \( v \). Finally, the line 35 outputs the paths constructed by Algorithm 5. In this section, we use \( P_i \) to denote the path set obtained by Algorithm 5.

For \( 1 \leq i \leq r \) and \( i \neq k \), we let the starting bit be \( i \), and let \( P_i' \) represent the set of all paths with starting bit \( i \). Algorithm 5 will call Algorithm 1 (PA). If we start a bit-changing process from bit \( i \), the range of the value of \( u_i \) is \([0, m_i - 1]\) \( \{ u_i \} \).

**Algorithm 5:** BP2--4\((u, v, x, y, G, k, n, L_w, l, r)\)

Input: vertices \( u, v, x, y, \) the graph \( G \) and the array \( L_w \), and indexes \( k, n, l, r \).

Output: disjoint paths from \( u \) to \( v \) which all pass through \( x \) and \( y \).

begin

\[ P_i \leftarrow \emptyset, \quad a = |L_w|, \quad s_1 = s_2 = 0, \quad a_s = a_1 = a_2 = s_2, \]
\[ n_0 = x', \quad n_1 = (n_0)^{a_1}_{a_1'}, \quad n_2 = (n_0)^{a_2}_{a_2'}; \]

for \( i = 1 \) to \( r \) do

\[ \text{for } j = 0 \text{ to } m_i - 1 \text{ do} \]

\[ \text{if } i \neq k \text{ and } u_i \neq j \text{ then} \]

\[ \text{if } k \in L_w \text{ then} \]

\[ \text{let } s_1 \text{ be the index such that } l_s = k; \]

end if

\[ \text{if } i \in L_w \text{ then} \]

\[ \text{let } s_2 \text{ be the index such that } l_s = i; \]

end if

\[ \text{if } s_1 \neq 0 \text{ and } s_2 \neq 0 \text{ then} \]

\[ a_1 = a_2 = a_1, \quad b_1 = a, \quad b_2 = b_3 = 0; \]

end if

\[ \text{if } s_1 = 0 \text{ and } s_2 > 0 \text{ then} \]

\[ a_1 = a_2 = 1, \quad a_3 = 0, \quad b_1 = a, \quad b_2 = s_2 - 1, \]
\[ b_3 = 0; \]

end if

end for

end for

end for

end for

end for

end

Therefore, the number of paths in \( P_i \) is \( m_i - 1 \). The number of paths in \( P_i \) is \( \sum_{i=1}^{r} (m_i - 1) = \kappa(G) - m_i + 1 \).

Then we will see how the algorithm works by going over an example. We set the generalized hypercube is \( G(4, 4, 4, 4, 4) \). Let \( u = 00000, v = 00111, x = 00020 \), and \( y = 00101 \). Then \( L_w = \{ 3, 2, 1 \} \). We have \( k = 2 \), \( n = 2 \), and \( k, n \in L_w \). According to the algorithm, we can construct paths with the value of \( i \) from 1 to 5 and \( i \neq 2 \). We know that when \( i \in L_w \) the values of \( i \) are 1, 2, and 3 when \( i \notin L_w \) the values of \( i \) are 4, 5. Then we can get the path sets as follows:

\[ P_1 = \{ (00000 \rightarrow 00020 \rightarrow 0002 j \rightarrow 0012 j \rightarrow 0010 j \rightarrow 00101 \rightarrow 00111) \mid 0 \leq j \leq m_1, j \neq u_1 \} \]
\[ = \{ (u, x, x_2', (x_1')^2, P(A((x_1')^2)_{y, L_w}, 0, 0, 0, 0, 0), y, v) \mid 0 \leq j \leq m_1 - 1, j \neq u_1 \}. \]

\[ P_2 = \{ (00000 \rightarrow 0020 \rightarrow 002j \rightarrow 00j21 \rightarrow 00j31 \rightarrow 00131 \rightarrow 00111) \mid 0 \leq j \leq m_1, j \neq u_1 \} \]
\[ = \{ (u, x, x_2', (x_1')^2, P(A((x_1')^2)_{y, L_w}, 0, 0, 0, 0, 0), y, v) \mid 0 \leq j \leq m_1 - 1, j \neq u_1 \}. \]

\[ P_3 = \{ (00000 \rightarrow 00020 \rightarrow 00j21 \rightarrow 00j31 \rightarrow 00131 \rightarrow 00111) \mid 0 \leq j \leq m_1, j \neq u_1 \} \]
\[ = \{ (u, x, x_2', (x_1')^2, P(A((x_1')^2)_{y, L_w}, 0, 0, 0, 0, 0), y, v) \mid 0 \leq j \leq m_1 - 1, j \neq u_1 \}. \]

Therefore, the path set \( P_i = P_1 \cup P_2 \cup P_3 \cup P_4 \).
We will prove paths constructed by Algorithm 5 are disjoint. Then we have the following lemma.

**Lemma 5.** Algorithm 5 can construct $\kappa(G) - m_k + 1$ disjoint paths based on $u$ and $v$ in which each path passes through both $x$ and $y$.

**Proof.** We use $N_k$ to denote the number of paths in set $P_k$. Thus, $N_k = \sum_{i=1}^{r} (m_i - 1) - 1 = \kappa(G) - m_k + 1$. Then we set $P_i$ to be the $i$-th path set of $P_k$, where $1 \leq i \leq r$ and $i \neq k$. We let $Q_1$ and $Q_2$ be two different paths, where $Q_1, Q_2 \in P_k$. We will prove that paths $Q_1$ and $Q_2$ are disjoint. The proof is similar to Lemma 3. The lemma holds.

### 3.6 The Disjoint Paths Base on any Two Distinct Vertices in the Generalized Hypercube

In this section, we will use the four kinds of algorithms above mentioned to construct disjoint path based on any two distinct vertices in $G(m_1, m_2, ..., m_r)$. According to GHDP, we know that the complexity of these two algorithms is $O(mr)$, where $m$ is the cardinal of each dimension and $r$ is the dimension of the generalized hypercube. The maximum length of these paths constructed by four algorithms is $r + 2$. Then, we have the following theorem.

**Algorithm 6: GHDP($u, v, x, y, G, k, n, L_{uv}, l, r$)**

| Input: vertices $u$, $v$, $x$ and $y$, the graph $G$ and the array $L_{uv}$ and indexes $k, n, l, r$. |
| Output: disjoint paths from $u$ to $v$. |
| begin |
| $P \leftarrow \emptyset$ |
| if $n = k$ && $x_i = y_o$ then |
| $P = BP2 - 1 \cup BP2 - 4$ |
| else |
| $P = BP2 - 2 \cup BP2 - 3$ |
| end if |
| return $P$ |
| end |

**Theorem 4.** Algorithm 6 can construct at least $\kappa^1(G)$ disjoint paths based on any two distinct vertices in $G(m_1, m_2, ..., m_r)$ under 1-restricted connectivity.

**Proof.** Algorithm 6 constructs disjoint paths based on any two distinct vertices $u$ and $v$ in $G(m_1, m_2, ..., m_r)$ under 1-restricted connectivity. Obviously, we use BP2-1 and BP2-4 as a combination, BP2-2 and BP2-3 as a combination to construct the disjoint paths, respectively. According to Lemma 2 and Lemma 4, the numbers of disjoint paths constructed by BP2-1 and BP2-3 are both $\kappa(G) - 1$. According to Lemma 3 and Lemma 5, the numbers of disjoint paths constructed by BP2-2 and BP2-4 are both $\kappa(G) - m_k + 1$. In Algorithm 6, GHDP calls the two algorithms in different cases. Then according to Theorem 3, the number of disjoint paths constructed by Algorithm 6 is at least $\kappa^1(G)$. 

### 4 Simulation

In this section, we do simulations to analyze the performances of GHDP and compare it to other one existing algorithm. The simulation experiments are based on the eclipse tool. Then to avoid the occasionally of experimental results, every data we gotten in the experiment is the average of running 60 times. There are two parameters to evaluate the performances of algorithms: PN (The Number of Disjoint Paths) and FPN (The Number of Fault-tolerance Disjoint Paths). In [30], Tong et al. introduced an algorithm to construct disjoint paths in the generalized hypercube. However, the algorithm in [30] did not consider the restricted connectivity and it only considered the connectivity of the generalized hypercube. We called the algorithm proposed by Tong et al. to be ADP. Then we compare PN and FPN of GHDP and ADP, respectively.

#### 4.1 The Number of Disjoint Paths (PN)

There we use the PN indicator to estimate two algorithms in different $G(m_1, m_2, ..., m_r)$. For simplicity, we let $m_1 = m_2 = ... = m_r$.

Then we run these two algorithms in two groups of graphs. The first group of graphs is $G(5, 5, 5, 5)$, $G(6, 6, 6, 6)$, $G(7, 7, 7, 7, 7)$, $G(8, 8, 8, 8, 8, 8)$, and $G(9, 9, 9, 9, 9, 9)$. The second group of graphs is $G(7, 7, 7, 7)$, $G(7, 7, 7, 7, 7)$, $G(7, 7, 7, 7, 7, 7)$, $G(7, 7, 7, 7, 7, 7, 7)$, and $G(7, 7, 7, 7, 7, 7, 7, 7)$. The results are shown in Figure 3.

**Figure 3.** The comparation of GHDP and ADP about PN

In Figure 3(a), the abscissa denotes the number of vertices in each dimension of $G(m_1, m_2, ..., m_r)$ and $r = 5$. In Figure 3(b), the abscissa represents the value of $r$ of $G(m_1, m_2, ..., m_r)$ and the number of vertices in each dimension is 7. Based on 3(a) and 3(b), we know...
that the number of disjoint paths constructed by GHDP is much larger than that constructed by ADP. For example, GHDP can construct 35 disjoint paths and ADP can only construct 20 disjoint paths in \( G(5, 5, 5, 5, 5) \); GHDP can construct 65 disjoint paths and ADP can only construct 36 disjoint paths in \( G(7, 7, 7, 7, 7) \); GHDP can construct 80 disjoint paths and ADP can only construct 45 disjoint paths in \( G(10, 10, 10, 10, 10) \). The larger the graph dimension, the greater the performance difference between the two algorithms.

4.2 The Number of Fault-tolerance Disjoint Paths (FPN)

In this section, we assume that there are many faulty vertices in \( G(m, m, \ldots, m) \). Then, we use GHDP and ADP to construct fault-free disjoint paths in the generalized hypercube. We use \( N \) to represent the number of faulty vertices. Let the value of \( N \) be 20, 60, 180, 240, 300, and 360, respectively. In simulation, we assume that the distribution of faulty vertices in random. Then we run these two algorithms in four graphs: \( G(6, 6, 6, 6, 6) \), \( G(7, 7, 7, 7, 7) \), \( G(8, 8, 8, 8, 8) \), and \( G(10, 10, 10, 10, 10) \), respectively. Finally, we compared the number of fault-free disjoint paths constructed by GHDP and ADP (as shown in Figure 4). The value of FPN in Figure 4 is the average number of fault-free disjoint paths based on any two distinct vertices in the generalized hypercube.

From four figures, we know that GHDP has stronger fault-tolerant ability than ADP. For example, when the number of faulty vertices is 240 in \( G(7, 7, 7, 7, 7) \), the number of fault-free disjoint paths constructed by GHDP based on any two distinct vertices is 51, and the number of fault-free disjoint paths constructed by ADP based on any two distinct vertices is only 29; when the number of faulty vertices based on any two distinct vertices is 360 in \( G(10, 10, 10, 10, 10) \), the number of fault-free disjoint paths constructed by GHDP based on any two distinct vertices is 78, and the number of fault-free disjoint paths constructed by ADP based on any two distinct vertices is only 44.

As a consequence, we can construct more disjoint paths by using GHDP than ADP. When there are many faulty vertices in one network, GHDP can construct more fault-free disjoint paths than ADP. Thus, GHDP has stronger fault-tolerant ability than ADP. The performance of GHDP is better than ADP.

5 Conclusion and Further Work

In this paper, we are the first to propose an algorithm GHDP to construct disjoint paths based on any two distinct vertices in GH under 1-restricted connectivity. We can construct at least \( \kappa'(G) \) disjoint paths in \( O(nr) \) time by GHDP. The maximum length of disjoint paths constructed by GHDP in \( G(m, m, \ldots, m) \) is bounded by \( r + 2 \). The study for constructing disjoint paths under the restricted connectivity can be made further. The algorithm to construct disjoint paths under other connectivity such as g-restricted connectivity with \( g \geq 2 \) and structure connectivity [31] has not been studied, which is a problem worth studying, especially in the deformation of the hypercube such as the twisted cube [12], the crossed cube [19], and the spined cube [32].

Acknowledgment

This work is supported by the National Natural Science Foundation of China (No. 61572337 and No. 61872257), the Natural Science Foundation of the Jiangsu Higher Education Institutions (No. 18KJA520009), the Application Foundation Research of Suzhou of China (No. SYG201653), and A Project Funded by the Priority Academic Program Development of Jiangsu Higher Education Institutions.
References


**Biographies**

**Guijuan Wang** received the B.S. and M.S. degrees in computer science from Qufu Normal University, in 2013 and 2016, respectively. She is currently a Ph.D. candidate in computer science at Soochow University. Her research interests include parallel and distributed systems, algorithms, and interconnection architectures.

**Jianxi Fan** received the B.S., M.S., and Ph.D. degrees in computer science from Shandong Normal University, Shandong University, and City University of Hong Kong, China, in 1988, 1991, and 2006, respectively. He is currently a professor of computer science in the School of Computer Science and Technology at Soochow University, China. He visited as a research fellow the Department of Computer Science at City University of Hong Kong, Hong Kong (October 2006–March 2007, June 2009–August 2009, June 2011–August 2011). His research interests include parallel and distributed systems, interconnection architectures, design and analysis of algorithms, and graph theory.

**Yali Lv** received the B.S., M.S., and Ph.D. degrees in computer science from Henan Normal University, Yunnan University, and Soochow University, China, in 2003, 2006, and 2018, respectively. Her current research interests include interconnection networks for parallel and distributed computing, graph theory, and algorithms.

**Baolei Cheng** received the B.S., M.S., and Ph.D. degrees in Computer Science from the Soochow University in 2001, 2004, 2014, respectively. He is currently an Associate Professor of Computer Science with the School of Computer Science and Technology at the Soochow University, China. His research interests include parallel and distributed systems, algorithms, interconnection architectures, and software testing.

**Shuangxiang Kan** received the B.S. degree in information and computing science from Changshu Institute of Technology in 2018. He is currently a master candidate in computer science at Soochow University. His research interests include parallel and distributed systems, algorithms, and graph theory.