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

Guijuan Wang[1], Jianxi Fan[1], Yali Lv[2], Baolei Cheng[1], Shuangxiang Kan[1]

[1] School of Computer Science and Technology, Soochow University, China
[2] Institute of Information Technology, Henan University of Chinese Medicine, China
{guijuan_wang, lvyali136}@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 $G(m_r, m_{r-1}, ..., m_1)$ to denote the r-dimensional generalized hypercube and $\kappa^1(G)$ to denote the 1-restricted connectivity of $G(m_r, m_{r-1}, ..., m_1)$. Then we design an algorithm to construct at least $\kappa^1(G)$ disjoint paths based on any two distinct vertices in $G(m_r, m_{r-1}, ..., m_1)$ 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 has 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 $\kappa^g(G)$ 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

*Corresponding Author: Jianxi Fan; E-mail: jxfan@suda.edu.cn
DOI: 10.3966/160792642019102006028

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 $O(NlogN)$ recursive algorithm to construct *n* independent spanning trees in *Möbius* cubes, 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 $G(m_r, m_{r-1}, ..., m_1)$ 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 $O(mr)$ 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.

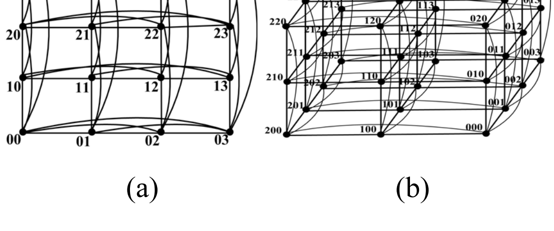
2.1 Definitions and Notations

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 $P(u^1, u^n) = (u^1 \rightarrow u^2 \rightarrow ... \rightarrow u^n)$ be a path from u^1 to u^n in which any two consecutive vertices are adjacent. Let $P(u^i, u^j) = (u^i \rightarrow u^{i+1} \rightarrow ... \rightarrow u^j)$. Then we call path $P(u^i, u^j)$ to be the sub-path of $P(u^1, u^n)$. Furthermore, we write $P(u^1, u^n) = (u^1 \rightarrow ... u^{i-1} \rightarrow u^{i+1} \rightarrow P(u^i, u^j) \rightarrow u^{j+1} ... \rightarrow u^n)$. We use *F* to represent the faulty vertices set of *G*. If a vertex $u \in F$, we call *u* a faulty vertex; otherwise we call it fault-free. If each vertex has one fault-free neighbor in graph *G-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 $G(m_r, m_{r-1}, ..., m_1)$, has $\prod_{i=1}^{r} m_i$ vertices, where m_i is the number of vertices in each dimension. Each vertex *u* in $G(m_r, m_{r-1}, ..., m_1)$ can be denoted by an *r*-digit identifier $u_r u_{r-1} ... u_1$, where $0 \leq u_i \leq m_i - 1$ with $1 \leq i \leq r$. Two vertices in $G(m_r, m_{r-1}, ..., m_1)$ 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 $G(m_r, m_{r-1}, ..., m_1)$ is made up of m_r $G(m_{r-1}, m_{r-2}, ..., m_1)$.



(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.

Theorem 1. The connectivity of $G(m_r, m_{r-1}, ..., m_1)$ is $\kappa(G) = \sum_{i=1}^{r} m_i$.

Theorem 2. The diameter of $G(m_r, m_{r-1}, ..., m_1)$ is *r*.

2.2 Properties of $G(m_r, m_{r-1}, ..., m_1)$ Under 1-restricted Connectivity

In this paper, we assume that each vertex in $G(m_r, m_{r-1}, ..., m_1)$ has at least one fault-free neighbor. For simplicity, let *G* represent $G(m_r, m_{r-1}, ..., m_1)$ in the following section and these two symbols can be used alternately. Given two arbitrary vertices $u = u_r u_{r-1} ... u_{k+1} u_k u_{k-1} ... u_1$ and $v = v_r v_{r-1} ... v_{n+1} v_n v_{n-1} ... v_1$ in $G(m_r, m_{r-1}, ..., m_1)$. Let $x = u_r u_{r-1} ... u_{k+1} g u_{k-1} ... u_1$ be a *u*'s fault-free neighbor, $u_k \neq g$; and $y = v_r v_{r-1} ... v_{n+1} q v_{n-1} ... v_1$ be a *v*'s fault-free neighbor, $v_n \neq q$. We use an array $L_{uv} = [l_\alpha, l_{\alpha-1}, ..., l_1]$ 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 $\{i \mid u_i \neq v_i\}$ [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 $\kappa^1(G)$ 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 $G(m_r, m_{r-1}, ..., m_1)$ is $\kappa^1(G) = 2\kappa(G) - n$, where $n = \max\{m_i \mid 1 \leq i \leq r\}$.

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).

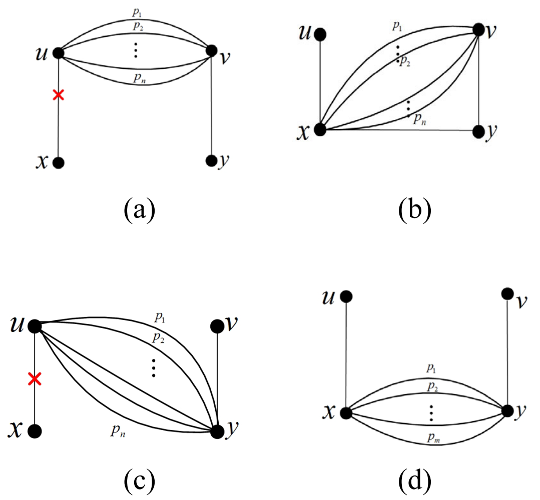


Figure 2. Four kinds of disjoint paths.

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*=0000, *v* = 0433, 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 α bit-changes, meaning the length of *P* is at least α . 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 $\rightarrow$ 000033 $\rightarrow$ 000433. (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_t . 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_t 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 z_t, $t = m_1 + m_2 + m_3$.
begin
  for $i = 1$ to 3 do
    if $s_i \neq 0$ then $m_i = |t_i - s_i| + 1$;
    else $m_i = 0$;
  end for
if $s_1 \neq 0$ then
    $z_i = (z_{i-1})^{v_{l_{s_1+i-1}}}_{l_{s_1+i-1}}$ for $1 \le i \le m_1$;
end if
if $s_2 \neq 0$ then
    $z_i = (z_{i-1})^{v_{l_{s_2+i-m_1-1}}}_{l_{s_2+i-m_1-1}}$ for $m_1 + 1 \le i \le m_1 + m_2$;
end if
if $s_3 \neq 0$ then
    $z_i = (z_{i-1})^{v_{l_{s_3+i-m_1-m_2-1}}}_{l_{s_3+i-m_1-m_2-1}}$,
    $m_1 + m_2 + 1 \le i \le m_1 + m_2 + m_3$;
end if
return $(z_0, z_1, ..., z_{m_1+m_2+m_3})$;
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_i^j to denote the vertex obtained by changing *u*'s *i*-th bit to *j*. For example, if *u*=010203, then $u_1 = 3$, u_1^4 = 010204, u_3 = 2, and u_3^4 = 010403. In Algorithm 1, when constructing the first sub-path, vertex $z_i = (z_{i-1})^{v_{l_{s_1+i-1}}}_{l_{s_1+i-1}}$ for $1 \le i \le m_i$. 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 $z_1 = (z_0)^{v_{l_{2+1-1}}}_{l_{2+1-1}}$, where $l_{2+1-1} = l_2 = 2$ and $v_{l_{2+1-1}} = v_2 = 1$, $z_1 = (z_0)^1_2$ = $(000000)^1_2$ = 000010. Similarly, $z_2 = (z_1)^{v_{l_{2+2-1}}}_{l_{2+2-1}}$ $= (z_1)^{v_{l_3}}_{l_3} = (000010)^{v_3}_3 = (000010)^1_3 = 000110$ and z_3 = $(z_2)^{v_{l_{2+3-1}}}_{l_{2+3-1}} = (z_2)^{v_{l_4}}_{l_4} = (000110)^{v_4}_4 = (000110)^3_4 = 003110$. The first sub-path is 000000 $\rightarrow$ 000010 $\rightarrow$ 000110 $\rightarrow$ 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, 110100 $\rightarrow$ 113100, and 113102 $\rightarrow$ 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 }= $\varnothing$, which denotes that the location of bit-changing is different for all vertices in each path. Therefore, vertices z_0, z_1, …, z_α 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_1 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 *x* are different, *r* is the dimension of the generalized hypercube, *G* represents $G(m_r, m_{r-1}, ..., m_1)$, 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_1 and deletes the path that passes through vertex *x*.

Note that we can start a bit-changing process from any bit. For $1 \le i \le r$, let the starting bit be *i*, and let P_1^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 $k, 1, r$.
Output: disjoint paths from u to v, which do not pass through x.
begin
  $P_1 \leftarrow \varnothing$, $\alpha = |L_{uv}|$, $s_1 = s_2 = t_2 = s_3 = t_3 = 0$, $t_1 = \alpha$.

```
    for i = 1 to r do
        for j = 0 to m_i − 1 do
            if u_i ≠ j then
                if i ∉ L_uv then
                    P_1 = P_1 ∪ {u, PA(u_i^j, v, L_uv, 1, α−1, 0, 0, 0, 0), v};
                else
                    let s_1 be the index such that l_{s_1} = i;
                    s_2 = 1, t_2 = s_1 − 1;
                    P_1 = P_1 ∪ {u, PA(u_i^j, v, L_uv, s_1+1, t_1, s_2, t_2, s_3, t_3), v};
                end if
            end if
        end for
    end for
if k ∈ L_uv then
    let s_1 be the index such that l_{s_1} = k; s_2 = 1, t_2 = s_1 − 1;
end if
return P_1 − {(u, PA(u_k^{x_k}, v, L_uv, s_1+1, t_1, s_2, t_2, s_3, t_3), 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^j is [0, $m_i - 1$] \{ u_i }. Therefore, the number of paths in P_1^i is $m_i - 1$. The number of paths in P_1 is $\sum_{i=1}^{r}(m_i - 1) = \kappa(G)$. 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:

$P_1^1 = \{(00000 \to 0000j \to 0001j \to 0011j \to 00111) \mid 0 \le j \le m_1 - 1, j \ne u_1\} = \{(u, PA(u_1^j, v, L_{uv}, 2, 3, 0, 0, 0, 0), v) \mid 0 \le j \le m_1 - 1, j \ne u_1\}$.

$P_1^2 = \{(00000 \to 000j0 \to 001j0 \to 001j1 \to 00111) \mid 0 \le j \le m_2 - 1, j \ne u_2\} = \{(u, PA(u_2^j, v, L_{uv}, 3, 3, 1, 1, 0, 0), v) \mid 0 \le j \le m_2 - 1, j \ne u_2\}$.

$P_1^3 = \{(00000 \to 00j00 \to 00j01 \to 00j11 \to 00111) \mid 0 \le j \le m_3 - 1, j \ne u_3\} = \{(u, PA(u_3^j, v, L_{uv}, 1, 2, 0, 0, 0, 0), v) \mid 0 \le j \le m_3 - 1, j \ne u_3\}$.

$P_1^4 = \{(00000 \to 0j000 \to 0j001 \to 0j011 \to 0j111 \to 00111) \mid 0 \le j \le m_4 - 1, j \ne u_4\} = \{(u, PA(u_4^j, v, L_{uv}, 1, 3, 0, 0, 0, 0), v) \mid 0 \le j \le m_4 - 1, j \ne u_4\}$.

$P_1^5 = \{(00000 \to j0000 \to j0001 \to j0011 \to j0111 \to 00111) \mid 0 \le j \le m_5 - 1, j \ne u_5\} = \{(u, PA(u_5^j, v, L_{uv}, 1, 3, 0, 0, 0, 0), v) \mid 0 \le j \le m_5 - 1, j \ne u_5\}$.

We know $k = 2$, then we need to delete the path that passes through *x*, *p* =(00000 → 00010 → 00110 → 00111). Then $P_1^2 = P_1^2 - p$. Therefore, the path set $P_1 = P_1^1 \cup P_1^2 \cup P_1^3 \cup P_1^4 \cup P_1^5$.

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

Lemma 2. For $1 \le i \le r$, 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_1 to represent the number of paths in P_1. Then $N_1 = \sum_{i=1}^{r}(m_i - 1) - 1 = \kappa(G) - 1$. We let Q_1 and Q_2 be two different paths, where $Q_1, Q_2 \in P_1$. We will prove that paths Q_1 and Q_2 are disjoint. There are two cases.

Case 1: For $1 \le i \le r$, $Q_1, Q_2 \in P_1^i$. 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 $v1_i \ne v2_i$. Thus, paths Q_1 and Q_2 are disjoint.

Case 2: For $1 \le i \ne w \le r$, $Q_1 \in P_1^i$ and $Q_2 \in P_1^w$. 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, $v1_i \ne v2_i$ and $v1_w \ne v2_w$. Therefore, paths Q_1 and Q_2 are disjoint.

Case 2.2: $i \in L_{uv}$, $w \notin L_{uv}$ or $i \notin L_{uv}$, $w \in L_{uv}$. 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, $v1_i \ne v2_i$. Therefore, paths Q_1 and Q_2 are disjoint.

Case 2.3. $i, w \in L_{uv}$. Since $i \ne w$, 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 $l_{p1} = i$ and $l_{p2} = w$. It is obvious that $v1_{l_{p1}} v1_{l_{p1}-1} \quad v1_{l_{p2}} v1_{l_{p2}-1} \ne v2_{l_{p1}} \quad v2_{l_{p1}-1} v2_{l_{p2}} v2_{l_{p2}-1}$. 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_2. In this section, we use P_2 to denote the path set obtained by Algorithm 3.

Algorithm 3: $BP2-2(u, v, x, G, k, L_{uv}, 1, r)$

Input: vertices u and v, the graph G and the array, L_{uv} and indexes $k, 1, r$.
Output: disjoint paths from u to v which all pass through x.
begin
  $P_2 \leftarrow \varnothing$, $\alpha = |L_{uv}|$, $s_1 = s_2 = a_1 = a_2 = 0$, $n_0 = x_i^j$,
  $n_1 = (n_0)_{l_{a_1+1}}^{v_{l_{a_1+1}}}$, $n_k = (n_1)_k^{v_k}$;
  for $i = 1$ to r do
    for $j = 0$ to $m_i - 1$ do
      if $k \in L_{uv}$ then
        let s_1 be the index such that $l_{s_1} = k$;
      end if
      if $i \in L_{uv}$ then
        let s_2 be the index such that $l_{s_2} = i$;
      end if
      if $i \neq k$ && $u_i \neq j$ then
        if $|L_{uv}| == 3$ && $k \in L_{uv}$ && $i == max$ $\{L_{uv} \setminus \{k\}\}$ then
          $l_m = \{L_{uv} \setminus \{k, i\}\}$;
          $P_2 = P_2 \bigcup \{u, x, n_0, PA((n_0)_k^{v_k}, L_{uv}, m, m, 0, 0, 0, 0), v\}$;
      else
        if s_1 ==0 then
        if s_2 ==0 then
          $a_1 = s_1,\ a_2 = a_3 = 0,\ b_1 = \alpha,\ b_2 = b_3 = 0;$
        else
          $a_1 = s_2$, $a_2 = 1$, $a_3 = 0$, $b_1 = \alpha$,
          $b_2 = s_2 - 1$, $b_3 = 0$;
        end if
       else
        if s_2 ==0 then
          $a_1 = s_2$, $a_2 = 1$, $a_3 = 0$, $b_1 = \alpha$,
          $b_2 = s_2 - 1$, $b_3 = 0$;
        end if
        if $s_1 > s_2$ then
          $a_1 = s_1$, $a_2 = 1$, $a_3 = s_2 + 1$, $b_1 = \alpha$,
          $b_2 = s_2 - 1$, $b_3 = s_1 - 1$;
        else
          $a_1 = s_1$, $a_2 = s_2 + 1$, $a_3 = 1$,
          $b_1 = s_2 - 1$, $b_2 = \alpha$, $b_3 = s_1 - 1$;
        end if
        $P_2 = P_2 \bigcup \{u, x, n_0, n_1, PA(n_k, L_{uv}, a_1 + 2,$
        $b_1, a_2, b_2, a_3, b_3), v\}$;
      end if
    end for
  end for
return P_2 ;
end

For $1 \le i \le r$ and $i \neq k$, we let the starting bit be i, and let P_2^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_2^i is $m_i - 1$. The number of paths in P_2 is $\sum_{i=1, i \neq k}^{r} (m_i - 1) = \kappa(G) - m_k + 1$.

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 the algorithm, we can construct paths with the value of i from 1 to 5 and $i \neq 2$. We know that when $i \in L_{uv}$ 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:

$P_2^1 = \{(00000 \rightarrow 00010 \rightarrow 0001j \rightarrow 0011j \rightarrow 00111) \mid 0 \le j \le m_1 - 1,\ j \neq u_1\} = \{(u, x, x_1^j, (x_1^j)_2^{v_2}, ((x_1^j)_2^{v_2})_3^{v_3}, PA((((x_1^j)_2^{v_2})_3^{v_3})_4^{v_4}, v, L_{uv}, 0, 0, 0, 0, 0, 0), v) \mid 0 \le j \le m_1 - 1,\ j \neq u_1\}$.

$P_2^3 = \{(00000 \rightarrow 00010 \rightarrow 00j10 \rightarrow 00j11 \rightarrow 00111) \mid 0 \le j \le m_3 - 1,\ j \neq u_3\} = \{(u, x, x_3^j, (x_3^j)_1^{v_1}, PA(((x_3^j)_1^{v_1})_4^{v_4} v, L_{uv}, 0, 0, 0, 0, 0, 0), v) \mid 0 \le j \le m_3 - 1, j \neq u_3\}$.

$P_2^4 = \{(00000 \rightarrow 00010 \rightarrow 0j010 \rightarrow 0j011 \rightarrow 0j111$ $00111) \mid 0 \le j \le m_4 - 1,\ j \neq u_4\} = \{(u, x, x_4^j,\ (x_4^j)_2^{v_2}, ((x_4^j)_2^{v_2})_1^{v_1}, PA((((x_4^j)_2^{v_2})_1^{v_1})_3^{v_3}, v, L_{uv}, 0,\ 0,\ 0,\ 0,\ 0,\ 0),\ v) \mid 0 \le j \le m_4 - 1, j \neq u_4\}$.

$P_2^5 = \{(00000 \rightarrow 00010 \rightarrow j0010 \rightarrow j0011 \rightarrow j0111$ $00111) \mid 0 \le j \le m_5 - 1,\ j \neq u_5\} = \{(u, x, x_5^j,\ (x_5^j)_2^{v_2}, ((x_5^j)_2^{v_2})_1^{v_1}, PA((((x_5^j)_2^{v_2})_1^{v_1})_3^{v_3}, v, L_{uv}, 0,\ 0,\ 0,\ 0,\ 0,\ 0),\ v) \mid 0 \le j \le m_5 - 1, j \neq u_5\}$.

Therefore, the path set $P_2 = P_2^1 \bigcup P_2^3 \bigcup P_2^4 \bigcup P_2^5$.

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_2. Thus, $N_2 = (\sum_{i=1}^{r} (m_i - 1)) - (m_k - 1) = \kappa(G) - m_k + 1$. Then we set P_2^i to be the i-th path set of P_2, where $1 \le i \le r$ and $i \neq k$. We let Q_1 and Q_2 be two different paths, where $Q_1, Q_2 \in P_2$. We will prove that paths Q_1 and Q_2 are disjoint. There are two cases.

Case 1: For $1 \le i \le r$ and $i \neq k$, $Q_1, Q_2 \in P_2^i$. 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 $1 \le i \neq w \le r$, $i \neq k$, $w \neq k$, $Q_1 \in P_1^i$ and $Q_2 \in P_1^w$. For any two different vertices $v1$ and $v2$

such that $v1 \in V(Q_1)$ and $v2 \in V(Q_2)$, 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, $v1_i \neq v2_i$ and $v1_w \neq v2_w$. 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, $v1_i \neq v2_i$. 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_3 to denote the path set obtained by Algorithm 4.

Algorithm 4: $BP2-3(u, v, y, G, n, L_{uv}, 1, r)$

Input: vertices u, v and y, the graph G and the array L_{uv}, and indexes $n, 1, r$.
Output: disjoint paths from u to v which all pass through x.
begin
 $P_3 \leftarrow \varnothing$, $\alpha = |L_{uv}|$, $s_1 = s_2 = 0$;
 for $i = 1$ to r do
  for $j = 0$ to $m_i - 1$ do
   if $n \in L_{uv}$ then
    let s_1 be the index such that $l_{s_1} = n$;
   end if
   if $i \in L_{uv}$ then
    let s_2 be the index such that $l_{s_2} = i$;
   end if
   if $i \neq n$ && $u_i \neq j$ then
    if s_1==0 && s_2==0 then
     $a_1 = a_2 = a_3 = 0$, $b_1 = \alpha$, $b_2 = b_3 = 0$;
    end if
    if s_1==0 && s_2>0 then
     $a_1 = s_2$, $a_2 = 1$, $a_3 = 0$, $b_1 = \alpha$, $b_2 = s_2 - 1$, $b_3 = 0$;
    end if
    if s_1>0 && s_2==0 then
     $a_1 = s_1$, $a_2 = 1$, $a_3 = 0$, $b_1 = \alpha$, $b_2 = s_1 - 1$, $b_3 = 0$;
    end if
    if $s_1 > s_2$ then
     $a_1 = s_1$, $a_2 = 1$, $a_3 = s_2 + 1$, $b_1 = \alpha$, $b_2 = s_2 - 1$, $b_3 = s_1 - 1$;
    else
     $a_1 = s_1$, $a_2 = s_2 + 1$, $a_3 = 1$, $b_1 = s_2 - 1$, $b_2 = \alpha$, $b_3 = s_1 - 1$;
    end if
    $P_3 = P_3 \cup \{u, u_i^j, PA((u_i^j)_n^{y_n}, L_{uv}, a_1+1, b_1, a_2, b_2, a_3, b_3)y, v\}$;
   end if
  end for
 end for
 return P_3;
end

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_3. For $1 \leq i \leq r$ and $i \neq n$, let this starting bit be *i*, and let P_3^i represent the set of all paths with starting 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_3^i is $m_i - 1$. The number of paths in P_3 is $\sum_{i=1}^{r}(m_i - 1) = \kappa(G)$. Then deleting the path that passes through *x*, the number of paths constructed by Algorithm 4 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* = 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:

$P_3^1 = \{(00000 \to 0000j \to 0002j \to 0012j \to 00121 \to 00111) \mid 0 \leq j \leq m_1 - 1,\ j \neq u_1\} = \{(u, u_1^j, PA((u_1^j)_n^{y_n}, v, L_{uv}, 3, 3, 1, 1, 0, 0), y, v) \mid 0 \leq j \leq m_1 - 1, j \neq u_1\}$.

$P_3^2 = \{(00000 \to 000j0 \to 001j0 \to 001j1 \to 00121 \to 00111) \mid 0 \leq j \leq m_2 - 1,\ j \neq u_2\} = \{(u, u_2^j, PA((u_2^j)_3^{v_3}, v, L_{uv}, 1, 1, 0, 0, 0, 0), y, v) \mid 0 \leq j \leq m_2 - 1, j \neq u_2\}$.

$P_3^3 = \{(00000 \to 00j00 \to 00j20 \to 00j21 \to 00121 \to 00111) \mid 0 \leq j \leq m_3 - 1,\ j \neq u_3\} = \{(u, u_3^j, PA((u_3^j)_n^{y_n}, v, L_{uv}, 1, 1, 0, 0, 0, 0), y, v) \mid 0 \leq j \leq m_3 - 1, j \neq u_3\}$.

$P_3^4 = \{(00000 \to 0j000 \to 0j020 \to 0j120 \to 0j121 \to 00121 \to 00111) \mid 0 \leq j \leq m_4 - 1,\ j \neq u_4\} = \{(u, u_4^j, PA((u_4^j)_n^{y_n}, v, L_{uv}, 3, 3, 1, 1, 0, 0), y, v) \mid 0 \leq j \leq m_4 - 1, j \neq u_4\}$.

$P_3^5 = \{(00000 \to j0000 \to j0020 \to j0021 \to j0121 \to 00121 \to 00111) \mid 0 \leq j \leq m_5 - 1,\ j \neq u_5\} = \{(u, u_5^j, PA((u_5^j)_n^{y_n}, v, L_{uv}, 3, 3, 1, 1, 0, 0), y, v) \mid 0 \leq j \leq m_5 - 1,\ j \neq u_5\}$.

We know $k = 2$, then we need delete the path that passes through *x*, $p = (00000 \to 00010 \to 00110 \to 00111)$. Then $P_3^2 = P_3^2 - p$. Therefore, the path set $P_3 = P_3^1 \cup P_3^2 \cup P_3^3 \cup P_3^4 \cup P_3^5$.

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_3 to denote the number of paths in set P_3. Thus, $N_3=(\sum_{i=1}^{r}(m_i-1))-1=\kappa(G)-1$. Then we set P_3^i to be the *i*-th path set of P_3, where $1\le i\le r$. We let Q_1 and Q_2 be two different paths, where $Q_1,Q_2\in P_3$. 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 all paths pass 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_4 to denote the path set obtained by Algorithm 5.

For $1\le i\le r$ and $i\ne k$, we let the starting bit be *i*, and let P_4^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: $BP2-4(u,v,x,y,G,k,n,L_{uv},1,r)$

Input: vertices u, v, x, and y, the graph G and the array L_{uv}, and indexes $k,n,1,r$.
Output: disjoint paths from u to v which all pass through x and y.
begin
  $P_4\leftarrow\varnothing$, $\alpha=|L_{uv}|$, $s_1=s_2=0$, $a_1=s_1$, $a_2=s_2$,
  $n_0=x_i^j$, $n_1=(n_0)_{l_{a_1+1}}^{v_{l_{a_1+1}}}$, $n_n=(n_1)_n^{y_n}$;
  for $i=1$ to r do
    for $j=0$ to m_i-1 do
      if $i\ne k$ && $u_i\ne j$ then
        if $k\in L_{uv}$ then
          let s_1 be the index such that $l_{s_1}=k$;
        end if
        if $i\in L_{uv}$ then
          let s_2 be the index such that $l_{s_2}=i$;
        end if
        if s_1==0 && s_2==0 then
          $a_1=a_2=a_3=0$, $b_1=\alpha$, $b_2=b_3=0$;
        end if
        if s_1==0 && s_2>0 then
          $a_1=s_2$, $a_2=1$, $a_3=0$, $b_1=\alpha$, $b_2=s_2-1$, $b_3=0$;
        end if
        if s_1>0 && s_2==0 then
          $a_1=s_1$, $a_2=1$, $a_3=0$, $b_1=\alpha$, $b_2=s_1-1$, $b_3=0$;
        end if
        if $s_1>s_2$ then
          $a_1=s_1$, $a_2=1$, $a_3=s_2+1$, $b_1=\alpha$, $b_2=s_2-1$, $b_3=s_1-1$;
        else
          $a_1=s_1$, $a_2=s_2+1$, $a_3=1$, $b_1=s_2-1$, $b_2=\alpha$, $b_3=s_1-1$;
        end if
        $P_4=P_4\cup\{u,x,n_0,n_1,PA(n_n,L_{uv},a_1+2,b_1,a_2,b_2,a_3,b_3)y,v\}$;
      end if
    end for
  end for
return P_4;
end

Therefore, the number of paths in P_4^i is m_i-1. The number of paths in P_4 is $\sum_{i=1,i\ne k}^{r}(m_i-1)=\kappa(G)-m_k+1$.

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 $L_{uv}=[3,2,1]$. We have $k=2$, $n=2$, and $k,n\in L_{uv}$. According to the algorithm, we can construct paths with the value of *i* from 1 to 5 and $i\ne 2$. 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:

$P_4^1=\{(00000\to 00020\to 0002j\to 0012j\to 0010j\to 00101\to 00111)\mid 0\le j\le m_1-1,\ j\ne u_1\}=\{(u,x,x_1^j,(x_1^j)_3^{v_3},PA(((x_1^j)_3^{v_3})_2^{y_2},v,L_{uv},0,0,0,0,0,0),y,v)\ |0\le j\le m_1-1,j\ne u_1\}$.

$P_4^3=\{(00000\to 0020\ \to 00j20\to 00j21\to 00j31\to 00131\to 00111)\mid 0\le j\le m_3-1,\ j\ne u_3\}=\{(u,x,x_3^j,(x_3^j)_1^{v_1},PA(((x_3^j)_1^{v_1})_2^{v_2}$ v, L_{uv}, 0, 0, 0, 0, 0, 0), y, v) $|0\le j\le\ |m_3-1,j\ne u_3\}$.

$P_4^4=\{(00000\to 00020\to 0j020\to 0j021\to 0j031\to 0j131\to 00131\to 00111)\mid 0\le j\le m_4-1,\ j\ne u_4\}=\{(u,x,x_4^j,(x_4^j)_1^{v_1},\ PA(((x_4^j)_1^{v_1})_2^{y_2},v,L_{uv},3,3,0,0,0,0),y,v)\,|0\le j\ \le m_4-1,j\ne u_4\}$.

$P_4^5=\{(00000\to 00020\to j0020\to j0021\to j0031\to j0131\to 00131\to 00111)\mid 0\le j\le m_5-1,\ j\ne u_5\}=\{(u,x,x_5^j,(x_5^j)_1^{v_1},\ PA(((x_5^j)_1^{v_1})_2^{y_2},v,L_{uv},3,3,0,0,0,0),y,v)\,|0\le j\ \le m_5-1,j\ne u_5\}$.

Therefore, the path set $P_4=P_4^1\cup P_4^3\cup P_4^4\cup P_4^5$.

We will prove paths constructed by Algorithm 5 are disjoint. Then we have the follow 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_4 to denote the number of paths in set P_4. Thus, $N_4=\sum_{i=1,i\neq k}^{r}(m_i-1)-1=\kappa(G)-m_k+1$. Then we set P_4^i to be the i-th path set of P_4, where $1\le i\le r$ and $i\neq k$. We let Q_1 and Q_2 be two different paths, where $Q_1, Q_2\in P_4$. 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 $G(m_r, m_{r-1},\ldots,m_1)$. 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: $GHDP(u,v,x,y,G,k,n,L_{uv},1,r)$

Input: vertices u, v, x and y, the graph G and the array L_{uv}, and indexes $k,n,1,r$.
Output: disjoint paths from u to v.
begin
  $P\leftarrow\varnothing$;
  if $n==k$ && $x_k==y_n$ 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 $G(m_r, m_{r-1},\ldots,m_1)$ under 1-restricted connectivity.

Proof. Algorithm 6 constructs disjoint paths based on any two distinct vertices u and v in $G(m_r, m_{r-1},\ldots,m_1)$ 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 $\kappa^1(G)$.

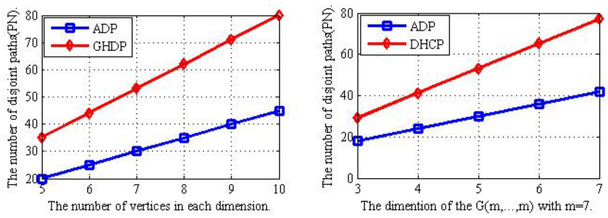
4 Simulation

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 $G(m_r, m_{r-1},\ldots,m_1)$. For simplicity, we let $m_r=m_{r-1}=\ldots=m_1$.

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



(a) (b)

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 $G(m_r, m_{r-1},\ldots,m_1)$ and $r=5$. In Figure 3(b), the abscissa represents the value of r of $G(m_r, m_{r-1},\ldots,m_1)$ 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, 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 $G(m_r, m_{r-1}, ..., m_1)$. 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, 120, 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.

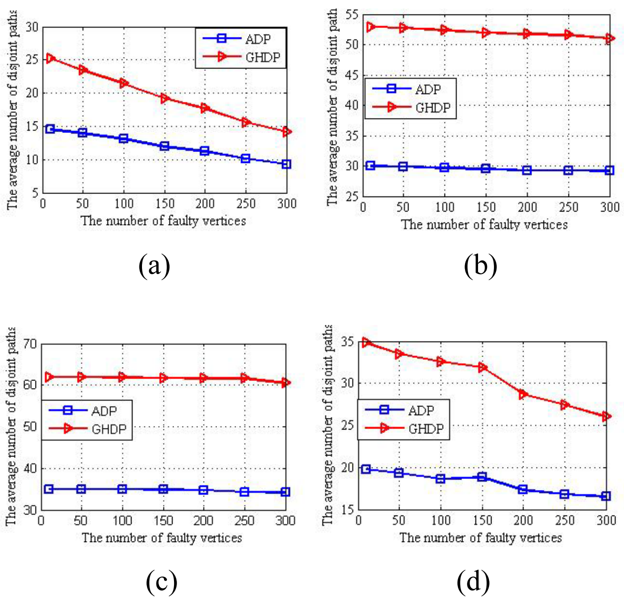


Figure 4. The comparation of GHDP and ADP about FPN

Base on Figure 4(a), 4(b), 4(c), and 4(d), we know that the larger the scale of the network, the smaller the impact of the faulty vertices on the construction of disjoint paths. Clearly, the more faulty vertices in one network, the fewer fault-free disjoint paths are constructed based on any two distinct vertices. For example, when the number of faulty vertices is 120 in G (6, 6, 6, 6, 6), the number of fault-free disjoint paths constructed by GHDP based on any two distinct vertices is at least 42; However, when the number of faulty vertices is 300 in G (6, 6, 6, 6, 6), the number of fault-free disjoint paths constructed by GHDP based on any two distinct vertices is 23.

From four figures, we know that GHDP has stronger fault-tolerance 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 $\kappa^1(G)$ disjoint paths in $O(mr)$ time by GHDP. The maximum length of disjoint paths constructed by GHDP in $G(m_r, m_{r-1}, ..., m_1)$ 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 of China (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

[1] L. N. Bhuyan, D. P. Agrawal, Generalized Hypercube and Hyperbus Structures for a Computer Network, *IEEE Transactions on Computers*, Vol. 33, No. 4, pp. 323-333, April, 1984.

[2] C. Guo, G. Lu, D. Li, H. Wu, X. Zhang, Y. Shi, C. Tian, Y. Zhang, S. Lu, BCube: A High Performance, Server-centric Network Architecture for Modular Data Centers, *Proceedings of the ACM SIGCOMM 2009 conference on Data communication*, Barcelona, Spain, 2009, pp. 63-74.

[3] J. H. Ahn, N. Binkert, A. Davis, M. McLaren, R. S. Schreiber, HyperX: Topology, Routing, and Packaging of Efficient Large-scale Networks, *Proceedings of the Conference on High Performance Computing Networking, Storage and Analysis*, Portland, Oregon, 2009, pp. 1-11.

[4] J. Kim, W. J. Dally, D. Abts, Flattened Butterfly: A Cost-efficient Topology for High-radix Networks, *ACM SIGARCH Computer Architecture News*, Vol. 35, No. 2, pp. 126-137, May, 2007.

[5] D. Li, J. Wu, On Data Center Network Architectures for Interconnecting Dual-port Servers, *IEEE Transactions on Computers*, Vol. 64, No. 11, pp. 3210-3222, November, 2015.

[6] K. Wada, T. lkeo, K. Kawaguchi, W. Chen, Highly Fault-tolerant Routings and Fault-Induced Diameter for Generalized Hypercube Graphs, *Journal of Parallel and Distributed Computing*, Vol. 43, No. 1, pp. 57-62, May, 1997.

[7] P. Fragopoulou, S. G. Akl, H. Meijer, Optimal Communication Primitives on the Generalized Hypercube Network, *Journal of Parallel and Distributed Computing*, Vol. 32, No. 2, pp. 173-187, February, 1996.

[8] S. G. Ziavras, Scalable Multifolded Hypercubes for Versatile Parallel Computers, *Parallel Processing Letters*, Vol. 5, No. 2, pp. 241-250, January, 1995.

[9] A. H. Esfahanian, S. L. Hakimi, On computing a conditional edge-connectivity of a graph, *Information Processing Letters*, Vol. 27, No. 4, pp. 195-199, April, 1988.

[10] A. H. Esfahanian, Generalized Measures of Fault Tolerance with Application to *n*-cube Networks, *IEEE Transactions on Computers*, Vol. 38, No. 11, pp. 1586-1591, November, 1989.

[11] Y. C. Chen, J. J. M. Tan, Restricted Connectivity for Three Families of Interconnection Networks, *Applied Mathematics & Computation*, Vol. 188, No. 2, pp. 1848-1855, May, 2007.

[12] S.-Y. Hsieh, H.-W. Huang, C.-W. Lee, {2,3}-restricted Connectivity of Locally Twisted Cubes, *Theoretical Computer Science*, Vol. 615, pp. 78-90, February, 2016.

[13] X. Wang, J. Fan, J. Zhou, C.-K. Lin, The Restricted *h*-connectivity of the Data Center Network DCell, *Discrete Applied Mathematics*, Vol. 203, pp. 144-157, April, 2016.

[14] X. Wang, J. Fan, X. Jia, C.-K. Lin, An Efficient Algorithm to Construct Disjoint Path Covers of DCell Networks, *Theoretical Computer Science*, Vol. 609, No. P1, pp. 197-210, January, 2016.

[15] C. Balbuena, X. Marcote, The *p*-restricted Edge-connectivity of Kneser Graphs, *Applied Mathematics and Computation*, Vol. 343, pp. 258-267, February, 2019.

[16] J. Fan, K. Li, S. Zhang, W. Zhou, B. Cheng, One-to-one Communication in Twisted Cubes under Restricted Connectivity, *Frontiers of Computer Science in China*, Vol. 4, No. 4, pp. 489-499, December, 2010.

[17] L. Lin, S.-Y. Hsieh, R. Chen, L. Xu, C.-W Lee, The Relationship between *g*-restricted Connectivity and g-good-neighbor Fault Diagnosability of General Regular Networks, *IEEE Transactions on Reliability*, Vol. 67, No.1, pp. 285-296, March, 2018.

[18] H. Lü, T. Wu, The Restricted *h*-connectivity of Balanced Hypercubes, *arXiv preprint* arXiv: 1805. 08461, May, 2018.

[19] S. Wang, X. Ma, The *g*-extra Connectivity and Diagnosability of Crossed Cubes, *Applied Mathematics and Computation,* Vol. 336, pp. 60-66, November, 2018.

[20] P. Li, M. Xu, Fault-tolerant Strong Menger (edge) Connectivity and 3-extra Edge-connectivity of Balanced Hypercubes, *Theoretical Computer Science,* Vol. 707, pp. 56-68, January, 2018.

[21] D.-W. Yang, Y.-Q. Feng, J. Lee, J.-X. Zhou, On Extra Connectivity and Extra Edge-connectivity of Balanced Hypercubes, *Applied Mathematics and Computation*, Vol. 320, pp. 464-473, March, 2018.

[22] X. Cai, E. Vumar, The Super Connectivity of Folded Crossed Cubes, *Information Processing Letters*, Vol. 142, pp. 52-56, February, 2019.

[23] K. Kaneko, S. Peng, Disjoint Paths Routing in Pancake Graphs, *IEEE International Conference on Parallel and Distributed Computing, Applications and Technologies (PDCAT)*, Taipei, Taiwan, 2006, pp. 254-259.

[24] C.-N. Lai, Optimal Construction of All Shortest Node-disjoint Paths in Hypercubes with Applications, *IEEE Transactions on Parallel and Distributed Systems*, Vol. 23, No. 6, pp. 1129-1134, June, 2012.

[25] B. Cheng, J. Fan, X. Jia, S. Zhang, B. Chen, Constructive Algorithm of Independent Spanning Trees on Möbius Cubes, *The Computer Journal*, Vol. 56, No. 11, pp. 1347-1362, November, 2013.

[26] D. Cheng, R.-X. Hao, Y.-Q. Feng, Two Node-disjoint Paths in Balanced Hypercubes, *Applied Mathematics and Computation*, Vol. 242, pp. 127-142, September, 2014.

[27] T. Inoue, Reliability Analysis for Disjoint Paths, *IEEE Transactions on Reliability*, Vol. 68, No. 3, pp. 985-998, September, 2019.

[28] L.-H. Hsu, C.-K. Lin, *Graph Theory and Interconnection Networks*, CRC Press, 2008.

[29] L. Guo, X. Wang, C.-K. Lin, J. Zhou, J. Fan, A Fault-free Unicast Algorithm in the Generalized Hypercube with Restricted Faulty Vertices, *International Journal of Foundations of Computer Science*, Vol. 28, No. 7, pp. 915-929, November, 2017.

[30] M. Tong, C. Liu, T. Fan, Routing Algorithms for Shortest Paths in Faulty Generalized Hypercubes, *Chinese Journal of Computers*, Vol. 21, No. 12, pp. 1074-1083, December, 1998.

[31] G. Wang, C.-K. Lin, B. Cheng, J. Fan, W. Fan, Structure Fault-tolerance of the Generalized Hypercube, *The Computer Journal*, Vol. 62, No. 10, pp. 1463-1476, October, 2019.

[32] W. Zhou, J. Fan, X. Jia, S. Zhang, The Spined Cube: A New Hypercube Variant with Smaller Diameter, *Information processing letters*, Vol. 111, No. 12, pp. 561-567, June, 2011.

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. He research interests include parallel and distributed systems, algorithms, and graph theory.