Cluster Computing (2024) 27:4347–4360
https://doi.org/10.1007/s10586-024-04582-5

Shortest node-to-node disjoint paths algorithm for symmetric networks

Hesham AlMansouri[1] · Zaid Hussain[2]

Received: 7 March 2024 / Revised: 26 April 2024 / Accepted: 17 May 2024 / Published online: 15 June 2024

Abstract
Disjoint paths are defined as paths between the source and destination nodes where the intermediate nodes in any two paths are disjoint. They are helpful in fault-tolerance routing and securing message distribution in the network. Several research papers were proposed to solve the problem of finding disjoint paths for a variety of interconnection networks such as Hypercube, Generalized Hypercube, Mesh, Torus, Gaussian, Eisenstein–Jacobi, and many other topologies. In this research, we have developed a general algorithm that constructs maximal node-to-node disjoint paths for symmetric networks where all paths are shortest. The algorithm presented in this paper outperforms other algorithms in finding not only the disjoint paths but shortest and maximal disjoint paths with a complexity of $O(n^2)$. In addition, we have simulated the proposed algorithm on different networks. The solution of unsolved problem in Cube-Connected-Cycles is given in the simulation results.

Keywords Interconnection network · Symmetric network · Edge disjoint · Disjoint paths · Fault-tolerant · Routing · Node-to-node

1 Introduction

Parallel computing is the computation of a program to perform tasks simultaneously. It is an essential field in computer science that advances the computation efficiency. There are two memory architectures in parallel computing, which are shared memory and distributed memory [1]. In shared memory, processors have access to and communicate through shared memories, whereas in distributed memory each processor has its own memory and all processors communicate by message passing. In distributed memory architecture, the processor and the memory are formed in a pair called Processing Element, PE [2]. Data is shared between processors in both memory architectures of parallel computing through an interconnection network. The work in this paper is based on the distributed memory architecture.

Interconnection networks play an essential role in the efficiency of parallel and distributed computing. One of the main characteristics of interconnection networks is their topology. There are many topologies existing in interconnection networks such as Hypercube [3], Mesh [4], k-ary n-cube [5], Cube-Connected Cycles [6], Torus [7], Gaussian [8, 9], Eisenstein–Jacobi [9, 10], and more can be found in this domain. The topology of an interconnection network determine how processors are connected. Each topology has characteristics such as degree, symmetry, regularity, diameter, and fault-tolerance. A symmetric network is a network that is both edge- and node-transitive [11]. Some application on interconnection networks are J-Machine [12], Cray T3D and T3E [13], and Gaussian cluster-based WSN [14].

Features of interconnection networks were and still are studied to improve the communication for data sharing in parallel computing. Many aspects are considered when sending messages from one node to another node. Routing is an essential communication method applied in computer

✉ Zaid Hussain
zhussain@cs.ku.edu.kw

Hesham AlMansouri
hmansouri@kisr.edu.kw

1 Systems and Software Development Department, Kuwait Institute for Scientific Research, Kuwait, Kuwait

2 Computer Science Department, Kuwait University, Sabah Al Salem University City, Kuwait

networks. There have been extensive researches in developing various routing algorithms in interconnection networks. Mostly, routing algorithm between source and destination nodes is provided with the proposed interconnection network. Further studies on routing is finding the disjoint paths between the given source and destination nodes. Disjoint paths play an important part in the efficiency of the message sharing in an interconnection network. The problem of finding disjoint paths can be classified into node-to-node, node-to-set, and set-to-set. In node-to-node, a message is sent from a source node to a destination node. In node-to-set, a message is sent from a source node to multiple destination nodes. In set-to-set, multiple source nodes send their messages to multiple destination nodes. Disjoint paths have several applications and benefits. For example, disjoint paths can improve throughput and load balance in a network [15, 16]. Another application is secured message routing, where disjoint paths can be used to ensure the security from eavesdropping in a network [17]. In addition, disjoint paths are used to ensure fault-tolerance in a network and make it more reliable. For instance, faulty nodes or edges in one path do not affect other disjoint paths, which makes the goal of delivering the message to the destination node achievable.

The main contributions of this work are as follows. (1) We provide a general algorithm that finds the disjoint paths in symmetric interconnection networks. (2) The proposed method is guaranteed to generate maximal shortest node-disjoint paths for a given network. (3) Providing the solution of an unsolved problem in Connected-Cube-Cycles. (4) Giving the results of applying the proposed algorithm on some interconnection networks.

Throughout this paper, we use the following words interchangeably: graph and network, vertices and nodes, and edges and links.

The paper is organized as follows: in Sect. 2 we review the related work. In Sect. 3, we demonstrate the required background used in this work. The general disjoint paths algorithms for symmetric networks is given in Sect. 4. Simulation results are shown in Sect. 5. Finally, the paper and its future work are concluded in Sect. 6.

2 Related work

In this section, we provide an overview of the existing research on disjoint paths algorithms for various interconnection network topologies. We briefly describe the key features and limitations of each approach, which are relevant to understanding the need for and contribution of our proposed algorithm.

Suurballe's algorithm, introduced in [18], is a popular approach for finding disjoint paths in a weighted directed graph. The algorithm works by first finding the shortest path from the source to the destination using Dijkstra's algorithm, then modifying the edge weights and searching for a second shortest path. While Suurballe's algorithm is efficient, it is limited to weighted directed graphs and does not guarantee the maximal shortest disjoint paths.

Bhandari proposed an algorithm in [19], to find disjoint paths in directed graphs with non-negative weights. Similar to Suurballe's algorithm, Bhandari's algorithm first computes the shortest path from the source to the destination using Dijkstra's algorithm. It then reverses the direction of the edges in the shortest path and applies Dijkstra's algorithm again. Although Bhandari's algorithm can find disjoint paths, maximal shortest disjoint paths between two nodes are not guaranteed.

The Ford–Fulkerson algorithm, originally designed for solving the maximum flow problem in a flow network [20], can be adapted for finding disjoint paths between a source and a destination node. By considering the paths as flows in the network and the edges as having unit capacities, the algorithm can identify the maximum number of edge-disjoint paths. However, the Ford–Fulkerson algorithm finds edge-disjoint paths, which may not necessarily be node-disjoint, and it does not guarantee the shortest paths. Moreover, its time complexity is $O(|V|^2|E|)$, where $|V|$ is the number of nodes in the network and $|E|$ is the number of edges in the network, where our proposed algorithm can outperform the complexity of the Ford–Fulkerson algorithm. Thereafter, while the Ford–Fulkerson algorithm can be adapted for finding disjoint paths, it may not be the most efficient or suitable approach for symmetric interconnection networks due to its limitations in providing node-disjoint paths, guaranteeing shortest paths, and its time complexity. This further emphasizes the need for an algorithm specifically designed to address the unique challenges of finding maximal shortest disjoint paths in symmetric interconnection networks.

Several algorithms were developed that are topology specific, where for each topology different steps are followed to get the disjoint paths between two nodes. There are various research works on parallel and disjoint paths for different topologies of interconnection networks. The node-to-node disjoint paths problem has been solved for Hypercube Network in [21–25]. Moreover, the problem was solved for Torus Network in [26, 27]. In addition, node-to-node disjoint routing for k-ary n-cube is presented in [5, 28–31]. Furthermore, the problem was solved for Hierarchical Hypercube, Dual-Cubes and WK Recursive in [32–34] respectively. Another variation of the disjoint paths problem is the node-to-set disjoint paths. This problem is solved for Folded Hypercube, Dual-Cube, Gaussian Network, and WK Recursive in [34–37] respectively.

Furthermore, the set-to-set disjoint paths is another variation of the disjoint paths problem. The set-to-set disjoint paths routing is presented for Dual Cubes in [38], Hypercube in [39, 40], Torus in [41], Torus-Connected Cycles in [42], and Hierarchical Hypercube in [43].

3 Background

In this section, we review the terminologies used in graph theory and then we briefly describe some interconnection networks used in this work. At the end of this section, we describe a mobile network protocol which is adopted in our proposed algorithm.

3.1 Terminology and definitions

A graph, or undirected graph, is an essential concept in graph theory to model interconnection networks. The network is represented as $G(V, E)$, where V is the set of nodes and E is the set of edges. Considering two connected nodes u and v in a network G, the *edge* between u and v is represented as (u, v). There are two types of edges in a network, which are regular and wraparound edges. The regular edges are the edges between two physically adjacent nodes, while the wraparound edges are the edges between two nodes that are not physically adjacent or two boundary nodes. A *path* in a network is a set of nodes that are connected through a sequence of edges, where it starts with a source node and it ends with a destination node. If the destination node is the source node, then it is called a *cycle*. In a network, two nodes are called *neighbors* if and only if there is an edge between them. An *intermediate* node is a node in a path that is neither the source nor the destination node. The *hop* count is the number of nodes passed in a path excluding the starting node and including the cycles. All path lengths are measured in terms of hop count. The *distance* between two nodes in a network is the length of the shortest path between them. The *degree* of a node in a network is the number of edges incident with that node.

A network is called *regular* when all the nodes have the same degree. The *diameter* of the network is distance between two farthest nodes (i.e., the longest path) in the network where the distance is minimized. Two nodes in a network are called *adjacent* (i.e., neighbor) if the distance between them is 1. In some interconnection networks, the distance between two nodes is measured by Hamming distance like in Hypercube network [3] and others are measured by Lee distance like in *k-ary n-cube* network [5]. The *Hamming distance* (D_H) between two nodes is the number of different digits between the two nodes' addresses. For example, given two nodes (122) and (131), then the Hamming distance between them is $D_H(122, 131) = 2$. Further, given two nodes' addresses $a_0, a_1, a_2, \ldots, a_n$ and $b_0, b_1, b_2, \ldots, b_n$, where $n+1$ is the length of both addresses and having *k-ary* alphabet. The *Lee distance* (D_L) is defined by the following formula:

$$D_L = \sum_{i=0}^{n} min(|a_i - b_i|, k - |a_i - b_i|).$$

Given a binary string, a *bit-flip* is the bit that changes. A network is called *connected* if there is a path between every pair of nodes in the network. A network *cut* is the separation of the network to several sub-networks that are disconnected, which is achieved by the removal of certain number of nodes. The network *connectivity* is the minimum number of nodes that form a network cut. A *reserved* node is a node that is used once and cannot be used again, where it is considered removed from the network.

3.2 Interconnection networks

In this subsection, we briefly illustrate the interconnection networks used in our work.

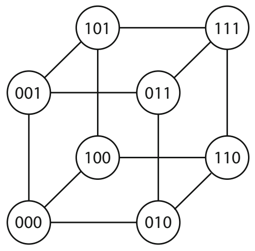
3.2.1 Hypercube

The Hypercube network is a popular topology in interconnection networks. It can be formed in many dimensions, for example, the 3D Hypercube is shown in Fig. 1. The nodes in the n-dimensional Hypercube are labeled in binary from 0 to $2^n - 1$, such that the Hamming distance between any two adjacent nodes is 1. In a Hypercube with N nodes, each node has a degree of $\log_2 N = n$ and the diameter of the network is $\log_2 N = n$. The distance between two nodes in the Hypercube is the Hamming distance between the addresses of the nodes.

3.2.2 2D torus

The 2D Torus ($T_{m \times n}$) network consisting of N nodes can be illustrated as $N = m \times n$, where m is the number of nodes along x-axis and n is the number of nodes along y-axis. Each node in the network has degree 4, i.e., Torus is 4-regular network. All boundary nodes of the Torus topology are connected to other boundary nodes on the same axis with wraparound links, and the corner nodes are connected

Fig. 1 3-Dimensional hypercube



to the adjacent corners of the x and y axis with wraparound links. Figure 2 shows an example of $T_{4\times4}$. The diameter of the Torus topology is $\lfloor m/2\rfloor + \lfloor n/2\rfloor$. Moreover, every node t in the Torus topology is labeled as t_{xy}, where x is the location of the node on the x-axis and y is the location of the node along y-axis.

3.2.3 Gaussian

Gaussian networks were proposed in [8, 9]. They are based on quotient ring of Gaussian integers formed as $\mathbb{Z}[i] = x + yi$ where $i = \sqrt{-1}$. The network can be generated by $\alpha = a + bi \neq 0$, where $0 \leq a \leq b$, and represented as a graph $G_\alpha(V,E)$ such that $V = \mathbb{Z}[i]_\alpha$ is the set of nodes and $E = \{(\beta,\gamma) \in V \times V \mid (\beta - \gamma) \equiv \pm1, \pm i \; mod \; \alpha\}$ is the set of edges. Gaussian networks are 4-regular networks having a total number of nodes $N(\alpha) = a^2 + b^2$ and the nodes are labeled as $x + yi$. Two nodes A and B in the network are adjacent if and only if $(A - B) \mod \alpha = \pm1$ or $\pm i$. The diameter of the network is $k = b$ if $N(\alpha)$ is even or $k = b - 1$ if $N(\alpha)$ is odd. The wraparound link for the boundary nodes can be computed with mod operation after adding ±1 or $\pm i$ to the node address. Figure 3 illustrates the Gaussian network generated by $\alpha = 4 + 5i$ where the dotted lines are the wraparound links.

3.2.4 Eisenstein–Jacobi

Similar to Gaussian networks, Eienstein–Jacobi (EJ) networks were proposed in [10] and [9]. They are based on quotient ring of EJ integers formed as $\mathbb{Z}[\rho] = x + y\rho$ where $\rho = (1 + i\sqrt{3})/2$ and $i = \sqrt{-1}$. The network can be generated by $\alpha = a + b\rho \neq 0$, where $0 \leq a \leq b$, and represented as a graph $EJ_\alpha(V,E)$ such that $V = \mathbb{Z}[\rho]_\alpha$ is the set of nodes and $E = \{(\beta,\gamma) \in V \times V \mid (\beta - \gamma) \equiv \pm1, \pm\rho, \pm\rho^2 \mod \alpha\}$ is the set of edges. EJ networks are 6-regular networks having a total number of nodes $N(\alpha) = a^2 + b^2 + ab$ and the nodes are labeled as $x + y\rho$. Two nodes A and B in the network are adjacent if and only if $(A - B) \; mod \; \alpha = \pm1$, $\pm\rho$, or $\pm\rho^2$. EJ networks are called *dense* when they contain a maximum number of nodes at distance k where k is the diameter of the network. Thus, the diameter of the dense EJ network is $k = a$. The wraparound link for the boundary nodes can be computed with mod operation after adding ±1, $\pm\rho$, or $\pm\rho^2$ to the node address. Figure 4 illustrates the EJ network generated by $\alpha = 3 + 4\rho$ where the dotted lines are the wraparound links.

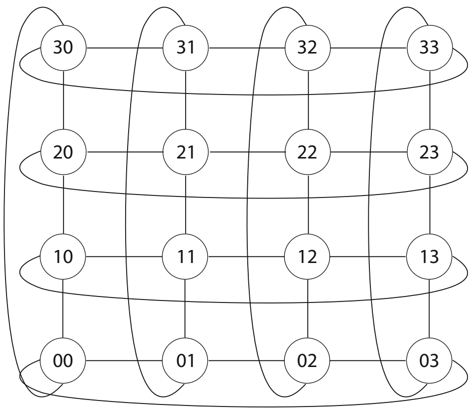


Fig. 2 2D torus network

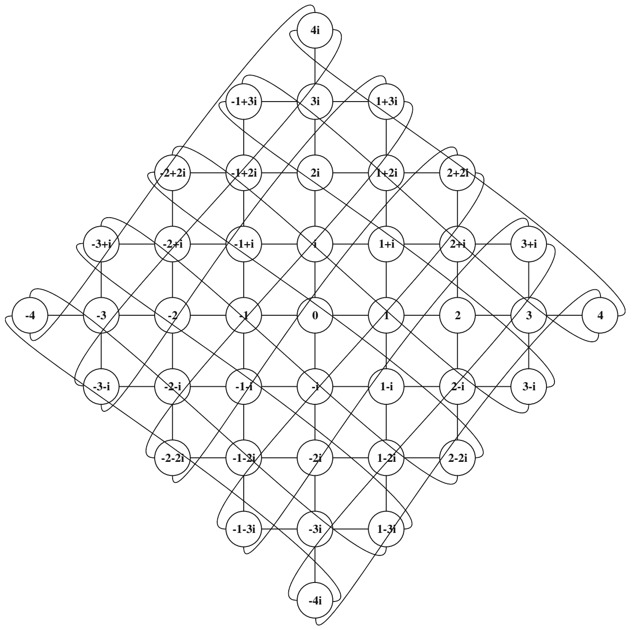


Fig. 3 Gaussian network generated by $\alpha = 4 + 5i$

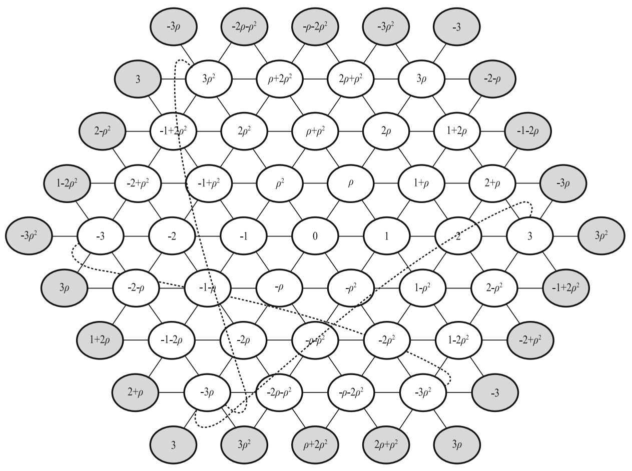


Fig. 4 EJ network generated by $\alpha = 3 + 4\rho$

3.2.5 Cube-connected cycles

The Cube-Connected Cycles is a topology similar to the Hypercube introduced in [6]. It is a multidimensional topology where the number of nodes is dependent on the number of dimensions. Specifically, the number of nodes in the Cube-Connected Cycles is $n2^n$ where n is the number of dimensions. Given an n-dimensional Hypercube, the Cube-Connected Cycles can be modeled by replacing each node in the Hypercube with a cycle of n nodes. The degree of the topology is 3, which is the main difference between the Cube-Connected Cycles and the Hypercube. The diameter of the topology is 6 for $n = 3$ and equal to $2n + \lfloor n/2 \rfloor - 2$ for $n \geq 4$ [44]. The nodes are labeled in the form of (x, y), where x is an integer from 0 to $2^n - 1$ and y is an integer from 0 to $n - 1$. Since each node is of degree 3, the neighbors of the node (x, y) are obtained by $(x, (y + 1)\ mod\ n)$, $(x, (y - 1)\ \mathrm{mod}\ n)$, and $(x\ XOR\ 2^y, y)$. Figure 5 illustrates the 3D Cube-Connected Cycles.

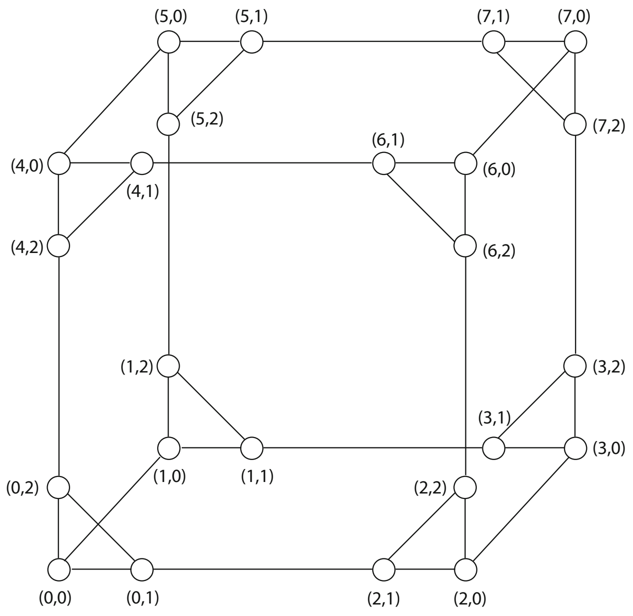


Fig. 5 3D cube-connected cycles

4 Disjoint paths algorithm

4.1 Overview

This algorithm is designed to find all distinct, non-overlapping paths within a network. It accomplishes this task through two stages: a detailed "micro" stage and a comprehensive "macro" stage. Each stage is built to efficiently traverse the network and identify individual paths extending from a start point (the source) to an end point (the destination). On a more detailed scale, the "micro" level begins at the source node and initiates a traversal of the network. As it moves forward, the algorithm logs the nodes it has visited and examines the neighbors of each node in sequence. A key feature of this algorithm is its ability to "reserve" nodes that form part of an identified path, ensuring each discovered path remains distinct as shown in Fig. 8. Simultaneously, the algorithm uses a specialized method to filter out any duplicate paths as illustrated in Fig. 6. This method, which employs recursion, compares each pair of paths to determine if they share nodes, ultimately generating a set of paths where the only common nodes are the source and destination as shown in Fig. 7. On a wider scale, the "macro" level oversees multiple rounds of these detailed traversals as demonstrated in Fig. 8. It keeps a count of the total number of unique paths identified and imposes a limit on the potential length of these paths. The algorithm also includes a mechanism for handling edge cases, such as a direct link between source and destination, to prevent it from becoming trapped in a loop. Contrary to certain protocols, this algorithm doesn't operate based on communication between nodes. Instead, it functions solely on the source node and uses a simulated version of the network. This is feasible due to our initial knowledge of the network topology, which differentiates it from networks that can undergo random structural changes. This characteristic improves the algorithm's efficiency and effectiveness in discovering unique paths.

The micro stage of the algorithm is similar to the well known breadth-first-search (BFS) algorithm. However, there are key differences between the both algorithms. Unlike BFS, which is primarily designed to locate a target node, our algorithm aims to identify all shortest disjoint paths between nodes in a network. This is achieved through a unique filtration process at each hop, which effectively reduces the complexity of selecting disjoint paths from exponential to polynomial. Additionally, our algorithm employs a dual-flag system for each node—'visited' and 'reserved'—which ensures the disjointness of paths across multiple executions, a feature not present in traditional BFS.

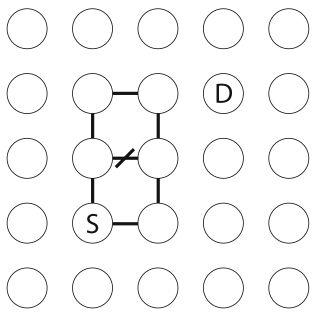


Fig. 6 Path filtration

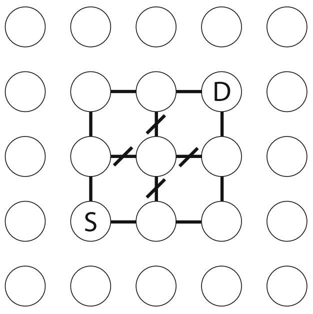


Fig. 7 Disjoint paths in the first macro iteration

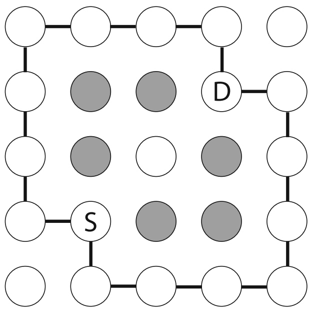


Fig. 8 Node reservation and second macro iteration

In summary, this algorithm provides a systematic and intelligent approach to identifying all possible unique paths within a network. This could be highly beneficial in scenarios such as network routing or communications, where the discovery of these paths could optimize system performance.

4.2 Proposed Algorithm

Algorithm 1 performs the macro discovery and it takes three parameters that are the network, source node, and destination node. The Algorithm 1 calls Algorithm 2 to perform the micro discovery, which resets the current node attributes except the nodes reserved.

Algorithm 2 takes the same parameters as Algorithm 1 and it also takes an empty queue in addition. Its major task is to filter the paths that reached the destination by calling Algorithm 5, and then it updates the disjoint paths list and reserves the nodes of the discovered disjoint paths. The algorithm starts by inserting the source node into the queue, then managing the queue until the queue becomes empty. The algorithm calls three other functions described in Algorithms 3 and 5.

Algorithm 3 takes the network, a node, and the queue as arguments and is used to discover the unvisited neighbors of the given node. The neighbor is considered unvisited if four conditions are met. The first two conditions are that the node must not be reserved, and its hop count must not exceed the network limit. The second two conditions are that the node must not be visited or has been visited in an equal hop count. The neighbor node is then flagged as visited and the hop count is set to the hop count of the sender incremented by one. In addition, all the paths of the sender have the neighbor node added to them and then are copied to be the paths to the neighbor. After that, if the neighbor is not the destination node then it is added to the queue. Finally, if the neighbor is the destination then set the network limit to its hop count.

Algorithm 4 is used to decide whether any two paths given of the same length are disjoint.

The disjoint paths filtering function in Algorithm 5 is responsible for returning the maximal disjoint paths in the iteration. It takes parameters an index equal to 0, an empty set of paths, and all the paths to destination that are of the same length to be compared against each other to generate the required paths.

Algorithm 1 Macro discovery

```
Input: $G, src, dest$
Output: none
1: if there exists an edge between $src$ and $dest$ then
2:     remove edge between $src$ and $dest$
3:     $disjointPaths \leftarrow append([src, dest])$
4: end if
5: while $disjointPaths.length < G.Degree$ do
6:     $microDiscovery(G, src, dest, queue \leftarrow [])$
7: end while
```

Algorithm 2 Micro discovery

```
Input: $G, src, dest, queue$
Output: none
1: $G' \leftarrow G$
2: $queue \leftarrow push(src)$
3: $src.hopCount \leftarrow 0$
4: $src.isVisited \leftarrow True$
5: $src.paths \leftarrow append(src)$
6: while $x \leftarrow queue.pop$ do
7:     $x.paths \leftarrow FDP(0, [], x.paths)$
8:     $addNeighbors(G', x, dest, queue)$
9: end while
10: if $dest.isVisited$ then
11:     $disjointPaths \leftarrow append(FDP(0, [], dest.paths))$
12:     for all $disjointPaths$ do
13:         for all nodes in $path$ do
14:             if $node \neq dest$ then
15:                 $node.isReserved \leftarrow True$
16:             end if
17:         end for
18:     end for
19: end if
```

4.3 Correctness of the algorithm

Let $G(V, E)$ be a symmetric network, where V is the set of nodes, and E is the set of edges. Let S and D be the source and destination nodes, respectively, and A be the queue of active nodes. Let $v \in V$, $N_d(v)$ be the set of neighboring nodes of v at distance d from the source node where $d \in \mathbb{Z}^+$, and $UN_d(v)$ be the set of unvisited neighboring nodes of v at distance d.

Lemma 1 *Every intermediate node is reached via the shortest available path from the originating source node S.*

Proof During the execution of the route discovery algorithm, each active node relays the route request to all its unvisited neighboring nodes. Since the route discovery process initiates from the source node S, all of S's immediate neighbors, denoted as $N_1(S)$, will receive this request first. Therefore, these nodes are reached at a distance $d = 1$, which is the minimum achievable distance from S.

For every node $u \in N_1(S)$, let's assume it becomes a new source node. We denote these new sources as $S_0, S_1, \ldots, S_m$ where $m = |N_1(S)|$, and m represents the total count of unvisited neighbors of the source node S.

Subsequently, each new source S_i for $0 \leq i \leq m$, propagates the route request to its own set of unvisited neighbors, which we will denote as $N_1(S_i)$. These neighboring nodes will be reached at a distance $d = 1$ from S_i, which is again the shortest possible distance from the respective S_i.

As we extend this process recursively, for any node $w \in N_l(S)$, it can be established that l is indeed the shortest distance from the original source node S due to the network symmetry constraint. This confirms that every intermediate node is reached via the shortest available path from the source node S. □

Algorithm 3 Add neighbors

```
Input: $G, node, dest, queue$
Output: none
1:  for all neighbors of node do
2:      if not $neighbor.isReserved$ and $node.hopCount < G.limit$ then
3:          if not $neighbor.isVisited$ or $neighbor.hopCount > node.hopCount$ then
4:              $neighbor.isVisited \leftarrow True$
5:              $neighbor.hopCount \leftarrow sender.hopCount + 1$
6:              $neighbor.paths \leftarrow sender.paths + neighbor$
7:              if neighbor $\neq$ dest and neighbor not in queue then
8:                  $queue \leftarrow append(neighbor)$
9:              end if
10:             if $neighbor = dest$ then
11:                 $G.limit \leftarrow neighbor.hopCount$
12:             end if
13:         end if
14:     end if
15: end for
```

Algorithm 4 Is Disjoint

```
Input: $p1, p2$
Output: $bool$
1: for $i = 0$ to $p1.length$ do
2:     if $p1[i] = p2[i]$ then
3:         return false
4:     end if
5: end for
6: return true
```

Lemma 2 *A node will not be reached by two paths of different lengths.*

Proof Let p be the distance a node x is reached. By Lemma 1, any other path to x of length k we have $k \geq p$. By Algorithm 3 line 3, $k \leq p$. Thus, by joining the two inequalities we get $p \leq k \leq p$. Therefore, $k = p$. □

Lemma 3 *In the route discovery process, a node v at a distance d from the source node S will not become active before another node u at a distance d' from S, where $d' < d$.* □

Proof The route discovery process initiates by propagating a route request from the source node S to its immediate neighbors. These neighbors are at a distance $d = 1$ from S. This initial step results in a set of active nodes, denoted as A, containing k nodes, where k equals the degree of the source node S.

In the subsequent steps of the process, each currently active node u identifies its unvisited neighbors, denoted as $UN_{d+1}(u)$, and appends them to the set A.

It is crucial to note that the activation of nodes within the set A adheres to a first-come-first-serve rule. In other words, the order of activation directly corresponds to the order of their inclusion in A.

Algorithm 5 Filter disjoint paths (FDP)

```
Input: $i, paths, remainingPaths$
Output: $disjointPaths$
 1: if $i = remainingPaths.length$ then
 2:     return $paths$
 3: end if
 4: $disjoint \leftarrow$ true
 5: for all path in paths do
 6:     if not $isDisjoint(remainingPath[i], path)$ then
 7:         $disjoint \leftarrow$ false
 8:         $break$
 9:     end if
10: end for
11: $withCurrent \leftarrow paths.copy()$
12: if $disjoint$ then
13:     $withCurrent.append(remainingPaths[i])$
14: end if
15: $withoutCurrent \leftarrow FDP(i + 1, paths, remainingPaths)$
16: $withCurrent \leftarrow FDP(i + 1, withCurrent, remainingPaths)$
17: if $withCurrent.length > withoutCurrent.length$ then
18:     return $withCurrent$
19: else
20:     return $withoutCurrent$
21: end if
```

Given this mechanism, it can be inferred that a node v at a distance $d + 1$ from S cannot be activated prior to a node u at a distance d from S. This is due to the fact that u would have been included in A before v, and as per the first-come-first-serve rule, u would be activated prior to v.

Therefore, the lemma is proved. □

Lemma 4 *A node will not initiate or broadcast a route request until it has been reached by all possible paths of equivalent length.*

Proof This statement can be supported by combining the conclusions from Lemmas 1, 2, and 3.

From Lemmas 1 and 2, it is established that during any given micro iteration, a node will not be reached by paths shorter or longer than the shortest path from the source node S. This implies that for a given node, all paths reaching it during a micro iteration will have identical lengths.

Additionally, Lemma 3 demonstrates that a node u at a distance d from the source S will not become active before a node v at a shorter distance d', where $d' < d$. This conclusion guarantees that nodes at a greater distance from the source are not prematurely activated, thus ensuring all shortest paths have time to reach the node.

With these considerations in mind, by the time a node becomes active and is ready to broadcast a route request, it would have been reached by all possible paths of equivalent, shortest length from the source. This confirms the statement in the lemma. □

Lemma 5 *A node does not rebroadcast route request in the same micro iteration.*

Proof Given that a node only broadcasts when it is active, i.e., it is a member of the queue of active nodes A. A node n with distance d from the source node can be in A if and only if it was an unvisited neighbor of u ($n \in UN_1(u)$), where u has shorter distance $d' < d$ from the source node. Since any unvisited neighbor of a node u in the network have greater distance than u and by Lemma 3, the node u will never be added to A if it was previously active. Therefore, a node u never rebroadcasts the route request. □

Lemma 6 *The algorithm will not generate non-disjoint paths.*

Proof Algorithm 5 compares all passed paths to generate all disjoint paths of the same length if they exist. Let R be the set of paths passed to Algorithm 5. Let $P_i = v_{i,0}, v_{i,1}, v_{i,2}, \ldots, v_{i,l}$ be a path and let $P_j = v_{j,0}, v_{j,1}, v_{j,2}, \ldots, v_{j,l}$ be another path such that $S = v_{i,0} = v_{j,0}$ and $D = v_{i,l} = v_{j,l}$, where l is the length of both paths such that $i \neq j$, and $P_i, P_j \in R$. For $0 \leq n \leq l$ and $0 \leq m \leq l$, if $v_{i,n} = v_{j,m}$ such that $v_{i,n} \neq S \neq D$ then by Lemma 2, $n = m$. In other words, if two paths are not disjoint, then the common intermediate nodes will be of the same distances from the source in both paths due to the network symmetry constraint. □

Lemma 7 *The algorithm will generate the maximal disjoint paths.*

Proof Let G be a symmetric graph with degree d and let S be the source node and D be the destination node. Let the shortest distance between S and D be L.

Suppose the algorithm finds p number of disjoint paths in the first iteration, which implies they are of length L by Lemma 2, where $p < d$. Thus, there must be $d - p$ unvisited neighbors of D due to network symmetry.

Consider an unvisited neighbor N of D. By definition of a shortest path, N must trivially be at distance at least L from the source, else N would have been visited.

N has $d - 1$ neighbors excluding D, where their distances from the source are at least $L - 1$. If all the neighbors of N are visited, this suggests their distances from the source are at most $L - 1$.

If a neighbor of N is used and has a distance L, it means it is the destination, and if it has a distance greater than L, it is outside the boundary of the paths of length L. Hence, if all neighbors of N are used, they have a distance exactly $L - 1$ from the source node.

Therefore, all the neighbors of N other than D are also neighbors of D. This contradicts the principles of network symmetry and Menger's theorem, because it suggests that there do not exist d disjoint paths from S to D.

By contradiction, then if all p paths are shortest, N must have an unvisited neighbor. This implies inductively that it remains connected to the graph and is reachable by S.

This lemma ultimately establishes that the algorithm will always find the maximal disjoint paths. □

Lemma 8 *The algorithm concludes its execution after at most d macro iterations, where d is the degree of the network.*

Proof Our objective is to show that the algorithm halts after no more than d macro iterations. A macro iteration is understood as a full cycle of finding and eliminating node-disjoint paths.

The continuation condition for the algorithm is the discovery of new node-disjoint paths. As per Lemma 7, the algorithm is designed to find the maximal set of node-disjoint paths between the source and destination nodes.

Consequently, as the degree d of the network signifies the maximum number of node-disjoint paths that could exist from any node, it serves as an upper bound for the number of macro iterations.

Once all possible disjoint paths are identified and eliminated, the algorithm ceases to continue as there are no new node-disjoint paths to be found. Therefore, the algorithm's termination is assured after at most d macro iterations, which aligns with the maximum degree of the network. □

Theorem 1 *The given algorithm is guaranteed to yield the maximal set of shortest node-disjoint paths between a specified source node and destination node within a network.*

Proof The proof of this theorem depends on the successful integration of several previously established and detailed lemmas, as discussed below.

From Lemma 1, we can infer that the algorithm ensures that each node in the network is reached via the shortest available path originating from the source node S. Simultaneously, Lemma 2 certifies that a node cannot be reached by two paths of differing lengths. As a result, we have a concrete basis that every node within the network is reached by the uniquely shortest path from the source node S.

Lemma 3 offers a significant contribution to the progression of the algorithm by establishing that the activation of nodes is directly proportional to their respective distances from the source node S. Complementarily, Lemma 4 asserts that a node will not initiate or broadcast a route request until it has been reached by all potential paths of equivalent length. Consequently, we can assert that every node becomes aware of all shortest paths leading to it before it propagates this information to its neighboring nodes.

Lemma 5 further refines the behavior of the algorithm by stating that a node does not rebroadcast a route request within the same micro iteration.

Importantly, Lemma 6 affirms that the algorithm will not generate non-disjoint paths, which is a crucial factor in maintaining the integrity and the goal of the process.

Furthermore, Lemma 7 bolsters the robustness of the algorithm by asserting its capacity to always identify the maximal number of node-disjoint paths.

Finally, Lemma 8 provides an essential termination guarantee, asserting that the algorithm will conclude in a finite number of steps, more precisely, within d macro iterations, with d representing the degree of the network.

In synthesis, the collective implications and assertions of these lemmas serve to confirm that the algorithm is proficient in identifying the maximal set of shortest node-disjoint paths between any designated source and destination nodes in the network. This cumulative reasoning corroborates the theorem. □

4.4 Complexity of the algorithm

The proposed algorithm for discovering all shortest disjoint paths in a symmetric network has five sub algorithms. The complexity for each algorithm is denoted as $\{A_1, A_2, A_3, A_4, A_5\}$ representing the Algorithm 1 to Algorithm 5 in Sect. 4.2. Let n be the number of nodes in the network, and d be the degree of the network, which is equal to the regularity, connectivity, and the maximum number of disjoint paths of the network. Let $l = n$ be the longest path in the network, $p = d^2$ be the maximum number of paths passed by the neighbours in the network, and $a = n$ be the maximum number of active nodes (queue size) in the network. The constant operations are denoted as c. The complexity of the algorithm is calculated in Table 1.

In order to assess the computational efficiency of the given algorithm, the complexity must be quantified. The algorithmic complexity is formulated as $n^2 * c$, in which n is indicative of the total nodes present within the network, and c serves as a constant derived from the equation 2^p, with p representing the maximum number of paths passed to the Algorithm 5 and is equal to d^2, where d is the degree of the network. When transposed into big O notation, which serves as a universally recognized means of expressing an algorithm's complexity, this calculation simplifies to $O(n^2)$. The rationale behind this simplification is predicated on the fact that d, the degree of the network, is a constant variable.

It is crucial to note that while the degree of most interconnection networks remains a relatively small number, the constant c can potentially escalate significantly in extreme scenarios of the algorithm. This escalation is particularly noticeable in networks of higher degrees, such as EJ network or the Hectagon network, where the potential for a higher number of intersecting nodes exists. Contrarily, in most pragmatic applications, the constant c remains low due to physical hardware limitations such as pin constraint [45], thereby optimizing the algorithm's efficiency. The reduced value of c can be attributed to the relatively infrequent occurrence of the worst-case scenario. This can also be justified by the fact that the number of paths passed to Algorithm 5 in a typical case is modest, as empirically validated by the data represented in Table 2.

It is, therefore, incumbent upon those implementing this algorithm to understand these dynamics thoroughly to maximize the computational efficiency while also being prepared for potential worst-case scenarios.

Table 1 Complexity

Algorithm	Complexity	Big O notation
A_1	$d \times (A_2) + c$	$O(n^2)$
A_2	$n \times (A_5 + A_3) + A_5 + d + c$	$O(n^2)$
A_3	$d + c$	$O(d)$
A_4	$n + c$	$O(n)$
A_5	$n \times p \times 2^p + c$	$O(n)$

5 Simulation results

This section discusses the practical implementation and tests of the algorithm in Python programming language using the NetworkX library. Several networks are tested, which are the 7×7 Torus, Gaussian (with $\alpha = 8 + 9i$, $\alpha = 8 + 11i$), EJ (with $\alpha = 8 + 9\rho$, $\alpha = 5 + 12\rho$ and $\alpha = 7 + 10\rho$), 5-dimensional Hypercube, and 4-dimensional Cube-Connected Cycles networks. In the following subsections, the source node is denoted as S and destination node is denoted as D. Due to the space limitations, all disjoint paths are summarized in Table 2, but the actual paths and figures that illustrate the paths are omitted and are provided in the GitHub repository.[1]

6 Conclusion and future work

There are many aspects for the efficiency of an interconnection network. One of the most important aspects is the topology of the network, where each topology has advantages over another. Discovering disjoint paths in interconnection networks differ between topologies. In this paper, an algorithm for discovering node-disjoint paths between two nodes in symmetric networks is presented. The general idea of the algorithm is that it traverse through unvisited neighbors of each node starting with the source node until the destination is reached. Then, all paths that reached the destination are filtered to return the disjoint paths of the same distance, if they exist. These paths are then reserved and omitted from the next repetitions, and then the algorithm repeats until the maximal disjoint paths are discovered. The key of control in the algorithm is having a queue that controls the order of nodes activated and to limit each node to be active only once. The algorithm runs on the source node only, since all the nodes have

[1] https://github.com/KISRDevelopment/Disjoint-Paths-Algorithm.

Table 2 Simulation

Topology	Source	Destination	iteration	Number of paths	Path length	Paths reached destination
5D Hypercube	00000	11001	1	3	3	6
			2	2	5	2
7×7 Torus	22	55	1	2	4	4
			2	2	7	2
	22	52	1	1	3	1
			2	1	4	1
			3	2	5	2
	33	44	1	2	2	2
			2	2	6	2
$\alpha = 8 + 9i$ Gaussian	0	$5 + 2i$	1	2	7	4
			2	2	10	4
	0	$3 + 3i$	1	2	6	4
			2	2	10	2
	0	5	1	1	5	1
			2	2	7	2
			3	1	12	1
$\alpha = 8 + 11i$ Gaussian	0	$8 + 4i$	1	2	12	4
			2	2	13	2
$\alpha = 8 + 9\rho$ EJ	0	$2 + 3\rho$	1	2	5	4
			2	2	7	2
			3	2	11	2
$\alpha = 8 + 9\rho$ EJ	0	$2 + 5\rho$	1	2	7	4
			2	2	9	2
			3	2	10	4
$\alpha = 8 + 9\rho$ EJ	0	5	1	1	5	1
			2	2	6	2
			3	2	9	2
			4	1	12	1
$\alpha = 5 + 12\rho$ EJ	0	$8 + \rho$	1	4	9	7
			2	2	11	4
$\alpha = 7 + 10\rho$ EJ	0	$8 + \rho$	1	6	9	9
4D CCC	(0, 0)	(0, 1)	1	1	1	0
			2	1	3	1
			3	1	7	1
	(0, 0)	(7, 3)	1	1	6	1
			2	1	8	1
			3	1	10	1
	(0, 0)	(14, 2)	1	2	7	4
			2	1	9	1

all the network information stored. The algorithm runs in a complexity of $O(n^2)$.

In this paper, the algorithm presented is limited to symmetric networks. Extending the algorithm to be more general and applying it on any network regardless of its symmetry is a research topic. The algorithm presented have a complexity of $O(n^2)$, but with further future work it may be possible to improve the performance and reduce the complexity. Further, a worthy problem is altering the algorithm to be run in a communication approach instead of a computational one. Finaly, the proposed algorithm discovers the node-to-node disjoint paths in symmetric networks. Whereas, developing a general algorithms for discovering the disjoint paths in node-to-set and set-to-set are left to be investigated.

There are no new data associated with this article. The source code of the simulation of this paper is available upon request.

Funding No funding.

Data availability Enquiries about data availability should be directed to the authors.

Declarations

Conflict of interest The authors have not disclosed any competing interests.

References

1. Jurczyk, M., Siegel, H.J., Stunkel, C.: Interconnection Networks for Parallel Computers, pp. 1613–1623. American Cancer Society, Atlanta (2009)
2. Kotsis, G.: Interconnection topologies and routing for parallel processing systems. Citeseer (1992)
3. Hayes, J.P., Mudge, T.: Hypercube supercomputers. Proc. IEEE **77**(12), 1829–1841 (1989)
4. Kumar, V., Grama, A., Anshul, G., Karypis, G.: Introduction to Parallel Computing: Design and Analysis of Algorithms, vol. 18, pp. 82–109. Benjamin/Cummings Publishing Company, San Francisco (1994)
5. Bose, B., Broeg, B., Kwon, Y., Ashir, Y.: Lee distance and topological properties of k-ary n-cubes. IEEE Trans. Comput. **44**(8), 1021–1030 (1995)
6. Preparata, F.P., Vuillemin, J.: The cube-connected-cycles: a versatile network for parallel computation. In: 20th Annual Symposium on Foundations of Computer Science (SFCS 1979), 1979, pp. 140–147. IEEE (1979)
7. Dally, W.J., Seitz, C.L.: The torus routing chip. Distrib. Comput. **1**(4), 187–196 (1986)
8. Martínez, C., Beivide, R., Stafford, E., Moretó, M., Gabidulin, E.M.: Modeling toroidal networks with the Gaussian integers. IEEE Trans. Comput. **57**(8), 1046–1056 (2008)
9. Flahive, M., Bose, B.: The topology of Gaussian and Eisenstein–Jacobi interconnection networks. IEEE Trans. Parallel Distrib. Syst. **21**(8), 1132–1142 (2009)
10. Martínez, C., Stafford, E., Beivide, R., Gabidulin, E.M.: Modeling hexagonal constellations with Eisenstein–Jacobi graphs. Probl. Inf. Transm. **44**(1), 1–11 (2008)
11. Holton, D.A., Sheehan, J.: The Petersen Graph, vol. 7. Cambridge University Press, Cambridge (1993)
12. Noakes, M.D., Wallach, D.A., Dally, W.J.: The j-machine multicomputer: an architectural evaluation. ACM SIGARCH Comput. Archit. News **21**(2), 224–235 (1993)
13. Scott, S.L., et al.: The Cray T3E Network: Adaptive Routing in a High Performance 3D Torus. Stanford University, Stanford (1996)
14. Quoc, D.N., Bi, L., Wu, Y., He, S., Li, L., Guo, D.: Energy efficiency clustering based on Gaussian network for wireless sensor network. IET Commun. **13**(6), 741–747 (2019)
15. Ganjali, Y., Keshavarzian, A.: Load balancing in ad hoc networks: single-path routing vs. multi-path routing. In: IEEE INFOCOM 2004, 2004, vol. 2, pp. 1120–1125 (2004). https://doi.org/10.1109/INFCOM.2004.1356998
16. Taft-Plotkin, N., Bellur, B., Ogier, R.: Quality-of-service routing using maximally disjoint paths. In: 1999 Seventh International Workshop on Quality of Service. IWQoS'99, 1999 (Cat. No. 98EX354), pp. 119–128 (1999). https://doi.org/10.1109/IWQOS.1999.766485
17. Bagchi, A., Chaudhary, A., Goodrich, M.T., Xu, S.: Constructing disjoint paths for secure communication. In: International Symposium on Distributed Computing, 2003, pp. 181–195. Springer (2003)
18. Suurballe, J.W., Tarjan, R.E.: A quick method for finding shortest pairs of disjoint paths. Networks **14**(2), 325–336 (1984). https://doi.org/10.1002/net.3230140209
19. Bhandari, R.: Optimal physical diversity algorithms and survivable networks. In: Proceedings of Second IEEE Symposium on Computer and Communications, 1997 (1997). https://doi.org/10.1109/iscc.1997.616037
20. Cormen, T.H., Leiserson, C.E., Rivest, R.L., Stein, C.: Section 26.2: The Ford-Fulkerson Method, 2nd edn., pp. 651–664. MIT Press, Cambridge (2001)
21. Lai, C.-N.: Optimal construction of all shortest node-disjoint paths in hypercubes with applications. IEEE Trans. Parallel Distrib. Syst. **23**(6), 1129–1134 (2011)
22. Sinanoglu, O., Karaata, M.H., AlBdaiwi, B.: An inherently stabilizing algorithm for node-to-node routing over all shortest node-disjoint paths in hypercube networks. IEEE Trans. Comput. **59**(7), 995–999 (2010)
23. Gu, Q.-P., Tamaki, H.: Routing a permutation in the hypercube by two sets of edge disjoint paths. J. Parallel Distrib. Comput. **44**(2), 147–152 (1997)
24. Madhavapeddy, S., Sudborough, I.H.: A topological property of hypercubes: node disjoint paths. In: Proceedings of the Second IEEE Symposium on Parallel and Distributed Processing 1990, 1990, pp. 532–539. IEEE (1990)
25. Gu, Q.P., Okawa, S., Peng, S.: Efficient Algorithms for Disjoint Paths in Hypercubes and Star Networks. Notes from the Institute of Mathematical Analysis (1994)
26. Day, K., Alzeidi, N., Arafeh, B., Touzene, A.: A parallel routing algorithm for torus NOCS. In: 2012 International Conference on Computer Networks and Communication Systems, 2012, vol. 35. Citeseer (2012)
27. Lai, C.-N.: Constructing all shortest node-disjoint paths in torus networks. J. Parallel Distrib. Comput. **75**, 123–132 (2015)
28. Xiang, Y., Stewart, I., Madelaine, F.: Node-to-node disjoint paths in k-ary n-cubes with faulty edges, In: 2011 IEEE 17th International Conference on Parallel and Distributed Systems, 2011, pp. 181–187. IEEE (2011)
29. Day, K., Al-Ayyoub, A.E.: Constructing node-disjoint routes in k-ary n-cubes. Sultan Qaboos Univ. J. Sci. **3**, 41–45 (1998)
30. Shih, Y.-K., Kao, S.-S.: One-to-one disjoint path covers on k-ary n-cubes. Theor. Comput. Sci. **412**(35), 4513–4530 (2011)
31. Zhuang, H., Li, X.-Y., Chang, J.-M., Lin, C.-K., Liu, X.: Embedding Hamiltonian paths in k-ary n-cubes with exponentially-many faulty edges. IEEE Trans. Comput. **72**(11), 3245–3258 (2023)
32. Wu, R.-Y., Chen, G.-H., Kuo, Y.-L., Chang, G.J.: Node-disjoint paths in hierarchical hypercube networks. Inf. Sci. **177**(19), 4200–4207 (2007)
33. Li, Y., Peng, S.: Fault-tolerant routing and disjoint paths in dual-cube: a new interconnection network. In: Proceedings. Eighth International Conference on Parallel and Distributed Systems. ICPADS 2001, 2001, pp. 315–322. IEEE (2001)

34. Bakhshi, S., Sarbazi-Azad, H.: One-to-one and one-to-many node-disjoint routing algorithms for WK-recursive networks. In: International Symposium on Parallel Architectures, Algorithms, and Networks (I-span 2008), 2008, pp. 227–232. IEEE (2008)
35. Lai, C.-N., Chen, G.-H., Duh, D.-R.: Constructing one-to-many disjoint paths in folded hypercubes. IEEE Trans. Comput. **51**(1), 33–45 (2002)
36. Kaneko, K., Peng, S.: Node-to-set disjoint paths routing in dual-cube. In: International Symposium on Parallel Architectures, Algorithms, and Networks (I-span 2008), 2008, pp. 77–82. IEEE (2008)
37. Alsaleh, O., Bose, B., Hamdaoui, B.: One-to-many node-disjoint paths routing in dense Gaussian networks. Comput. J. **58**(2), 173–187 (2015)
38. Kaneko, K., Peng, S.: Set-to-set disjoint paths routing in dual-cubes. In: 2008 Ninth International Conference on Parallel and Distributed Computing, Applications and Technologies, 2008, pp. 129–136. IEEE (2008)
39. Gu, Q.-P., Peng, S.: An efficient algorithm for set-to-set node-disjoint paths problem in hypercubes. In: Proceedings of 1996 International Conference on Parallel and Distributed Systems, 1996, pp. 98–105. IEEE (1996)
40. Park, J.-H., Kim, H.-C., Lim, H.-S.: Many-to-many disjoint path covers in hypercube-like interconnection networks with faulty elements. IEEE Trans. Parallel Distrib. Syst. **17**(3), 227–240 (2006)
41. Kaneko, K., Bossard, A.: A set-to-set disjoint paths routing algorithm in tori. Int. J. Netw. Comput. **7**(2), 173–186 (2017)
42. Bossard, A., Kaneko, K.: Set-to-set disjoint paths routing in torus-connected cycles. IEICE Trans. Inf. Syst. **99**(11), 2821–2823 (2016)
43. Bossard, A., Kaneko, K.: Set-to-set disjoint paths routing in hierarchical cubic networks. Comput. J. **57**(2), 332–337 (2014)
44. Friš, I., Havel, I., Liebl, P.: The diameter of the cube-connected cycles. Inf. Process. Lett. **61**(3), 157–160 (1997)
45. Dally, W.J., Towles, B.P.: Principles and Practices of Interconnection Networks. Elsevier, Amsterdam (2004)

Hesham AlMansouri received his B.Sc. degree in Computer Science from Gulf University for Science and Technology, Kuwait, in 2014, and his M.Sc. degree in Computer Science from Kuwait University, Kuwait, in 2020. He is currently a research associate at the Kuwait Institute for Scientific Research (KISR). Hesham's research interests encompass interconnection networks, machine learning, and cybersecurity. At KISR, he focuses on developing and implementing innovative machine learning models with applications across various domains. A significant aspect of his work involves collaborating with law enforcement agencies to build advanced machine learning models aimed at enhancing the effectiveness of cybercrime combating efforts.

Zaid Hussain received the B.Sc. degree in Computer Science from Kuwait University, Kuwait, in 2006; the M.Sc. and Ph.D. degrees in Computer Science from Oregon State University, USA, in 2008 and 2011, respectively. Since 2011, he has been with Kuwait University, Kuwait, where he is an associate professor with the Computer Science Department. His current research interests include parallel computing, interconnection networks, distributed systems, fault-tolerant computing.