ON STRONG FAULT TOLERANCE (OR STRONG MENGER-CONNECTIVITY) OF MULTICOMPUTER NETWORKS

A Dissertation

by

EUNSEUK OH

Submitted to the Office of Graduate Studies of
Texas A&M University
in partial fulfillment of the requirements for the degree of

DOCTOR OF PHILOSOPHY

August 2004

Major Subject: Computer Science

ON STRONG FAULT TOLERANCE (OR STRONG MENGER-CONNECTIVITY) OF MULTICOMPUTER NETWORKS

A Dissertation

by

EUNSEUK OH

Submitted to Texas A&M University
in partial fulfillment of the requirements
for the degree of

DOCTOR OF PHILOSOPHY

Approved as to style and content by:

Jianer Chen (Chair of Committee)	
Donald K. Friesen (Member)	Jennifer L. Welch (Member)
Gwan Choi (Member)	Valerie E. Taylor (Head of Department)

August 2004

Major Subject: Computer Science

ABSTRACT

On Strong Fault Tolerance (or Strong Menger-Connectivity) of Multicomputer Networks. (August 2004)

Eunseuk Oh, B.S., Hallym University, Korea;

M.S., Ewha Womans University, Korea

Chair of Advisory Committee: Dr. Jianer Chen

As the size of networks increases continuously, dealing with networks with faulty nodes becomes unavoidable. In this dissertation, we introduce a new measure for network fault tolerance, the *strong fault tolerance* (or *strong Menger-connectivity*) in multicomputer networks, and study the strong fault tolerance for popular multicomputer network structures. Let G be a network in which all nodes have degree d. We say that G is *strongly fault tolerant* if it has the following property: Let G_f be a copy of G with at most $d-2$ faulty nodes. Then for any pair of non-faulty nodes u and v in G_f, there are $\min\{deg_f(u), deg_f(v)\}$ node-disjoint paths in G_f from u to v, where $deg_f(u)$ and $deg_f(v)$ are the degrees of the nodes u and v in G_f, respectively.

First we study the strong fault tolerance for the popular network structures such as star networks and hypercube networks. We show that the star networks and the hypercube networks are strongly fault tolerant and develop efficient algorithms that construct the maximum number of node-disjoint paths of nearly optimal or optimal length in these networks when they contain faulty nodes. Our algorithms are optimal in terms of their time complexity.

In addition to studying the strong fault tolerance, we also investigate a more realistic concept to describe the ability of networks for tolerating faults. The traditional definition of fault tolerance, sustaining at most $d-1$ faulty nodes for a regular

graph G of degree d, reflects a very rare situation. In many cases, there is a chance that a routing path between two given nodes can be constructed though the network may have more faulty nodes than its degree. In this dissertation, we study the fault tolerance of hypercube networks under a probability model. When each node of the n-dimensional hypercube network has an independent failure probability p, we develop algorithms that, with very high probability, can construct a fault-free path when the hypercube network can sustain up to $2^n p$ faulty nodes.

To my husband Pablo and my son Josh

ACKNOWLEDGMENTS

First of all, I would like to express my profound gratitude to my advisor, Dr. Jianer Chen. His guidance and patience made this dissertation possible. His instruction and insight from the formative stages of the research through the final stages provided valuable direction to my work.

I am also thankful to the members of my committee, Dr. Donald K. Friesen, Dr. Jennifer L. Welch, and Dr. Gwan Choi for their encouragement and advice on my research, as well as Dr. David N. McMurray who served as the Graduate Counsel Representative on my committee.

Special thanks are due to Dr. Hongsik Choi and Dr. Hyeong-Ok Lee, who have given me their constant support and advice. I am also grateful to my friends, Dr. Tao Wang and Dr. Iyad Kanj. As friends and collaborators, they continuously encouraged and supported me in many ways and made my graduate life more enjoyable and memorable. I also gratefully acknowledge my friends Jong-Seok Kim and Seo-Young Ahn for their support and encouragement.

Most of all, thanks to my family who were always there when I needed them. They were always supportive throughout my research and gave me unconditional love. I dedicate this dissertation to my family.

TABLE OF CONTENTS

Page

LIST OF TABLES

LIST OF FIGURES

CHAPTER I

INTRODUCTION

A. Overview

Parallel computing has emerged because of the increasing size and complexity of computer problems. There is a large class of problems that can be broken down into smaller tasks and solved efficiently in parallel fashion such as image processing, modeling, and simulation. For such problems, the computational time can be significantly reduced by using a parallel computer which consists of processors that can work simultaneously on different parts of the problem. With increasingly faster and cheaper processors, parallel computers with a large number of processors have become feasible and realizable. The number of processors connected for a parallel computer can range from tens to several millions.

The topology of a parallel computer is called its interconnection network, which is often modeled as a graph where a processor is represented as a node, and a communication channel between processors is an edge between corresponding nodes. When an interconnection network is represented by such a graph, the fault tolerance is often measured by the vertex connectivity of the corresponding graph. The fault tolerance of the interconnection networks is the maximum number of nodes that can fail without preventing other non-faulty nodes from communicating. Formally speaking, the fault tolerance of a graph G is the maximum integer k that does not make G disconnect by removing any k nodes [2]. Thus, it is easy to observe that the fault tolerance of G is precisely one less than its connectivity. Menger [41] showed that if a graph G is d-connected, then every pair of nodes in G is connected by at least d node-disjoint

---

The journal model is *IEEE Transactions on Computers.*

paths. Assured by Menger's theorem, node-disjoint paths have been used to study the fault tolerance of interconnection networks.

With continuous increases in network size, routing in networks with faults has become unavoidable. Routing through node-disjoint paths between nodes can not only provide alternative routes to tolerate faulty nodes but also avoid communication bottlenecks. Moreover, routing through node-disjoint paths can speed up the transmission time by distributing data among disjoint paths. Thus, the study of disjoint paths connecting any two nodes can be useful for increasing the reliability of interconnection networks, as well as transmission efficiency. A larger number of disjoint paths is more desirable because of less vulnerability to disconnection.

The study of node-disjoint paths varies according to the number of source and destination nodes. There are three well-known paradigms: *one-to-one routing* that constructs the maximum number of node-disjoint paths in the network between two given nodes, *one-to-many routing* that constructs node-disjoint paths in the network from a given node to a given set of nodes, and *many-to-many routing* that constructs node-disjoint paths between a given set of nodes. Using these paradigms, node-disjoint paths have been extensively studied on networks [8, 18, 26]. Most research on constructing node-disjoint paths is done in graphs without faults. In this dissertation, we introduce the concept of strong fault tolerance which characterizes the property of parallel routing in a network with faulty nodes. We study this strong fault tolerance on popular interconnection networks such as the star networks and the hypercube networks. One of our goals is to develop algorithms that in optimal time, construct node-disjoint paths between two given nodes on interconnection networks with faulty nodes whose lengths are bounded by the shortest path length plus an additional small constant.

For a regular graph G of degree d, it can sustain at most $d-1$ faulty nodes to guarantee that all non-faulty nodes are connected. It is because a node in G will be disconnected from G if all neighbors of that node are removed. This traditional definition of fault tolerance for regular graphs reflects a very rare situation in that all neighbors incident on a node are faulty. In many cases, for two given nodes in a regular graph G, there is a chance that a routing path can be constructed between them though G may have more faulty nodes than its degree. Thus, the traditional definition of fault tolerance often underestimates the ability of networks to tolerate network faults; therefore, a more realistic concept to describe a network's ability to tolerate faults is needed. One is the probability model. For any two given nodes in a network, if with very high probability, we can construct a path between them, then the probability that the network is connected would be very high. We investigate the fault tolerance of hypercube networks under the probability model. In this research, our goal is to find routing algorithms that, with very high probability, construct a fault-free path between any two given nodes in hypercube networks, where hypercube networks can possibly be disconnected.

B. Strong Fault Tolerance

If two non-faulty nodes u and v of a given graph G are known, then we can easily detect the number of non-faulty neighbors of the nodes u and v. Based on this local information, constructing the maximum number of node-disjoint fault-free paths between them and analyzing the precise bound on the size of the number of faulty nodes allowed are interesting. Let G_f be a copy of a network G with a set S_f of faulty nodes, and u and v be non-faulty nodes in G_f. Then, we know $deg_f(u)$ and $deg_f(v)$, where $deg_f(u)$ and $deg_f(v)$ are the degrees of the nodes u and v in G_f. We are interested in constructing the maximum number of node-disjoint paths between

u and v in G_f. Obviously, the number of node-disjoint paths between u and v in G_f cannot be larger than $\min\{deg_f(u), deg_f(v)\}$. Motivated by this observation, we introduce a new measure for network fault tolerance: the *strong fault tolerance*.

Bound on the size of the faulty node set: We are interested in knowing the precise bound on the size of the faulty node set S_f such that for any two non-faulty nodes u and v in G_f, there are $\min\{deg_f(u), deg_f(v)\}$ node-disjoint paths between them. We observed that if the network G has all its nodes of degree d, then in general the number of faulty nodes in the set S_f should not exceed $d-2$ to ensure $\min\{deg_f(u), deg_f(v)\}$ node-disjoint paths between u and v in G_f. This can be seen as follows. Let u and v be two nodes in G whose distance is larger than 3. Pick any neighbor u' of u and remove the $d-1$ neighbors of u' that are not u. Note that no neighbor of u' can be a neighbor of v since the distance from u to v is at least 4. Let the resulting network be G_f. The degrees of the nodes u and v in G_f are d. However, there are obviously no d node-disjoint paths in G_f from u to v since one of the d neighbors of u in G_f, the u', leads to a "deadend". This motivates the following definition.

Definition A regular network G of degree d is *strongly fault tolerant* (or *strongly Menger-connected*) if for any copy G_f of G with at most $d-2$ faulty nodes, every pair of non-faulty nodes u and v are connected by $\min\{deg_f(u), deg_f(v)\}$ node-disjoint paths in G_f.

Strong fault tolerance characterizes the property of parallel routing in a network with faulty nodes. Since one of the motivations of network parallel routing is to provide alternative routing paths when failures occur, strong fault tolerance can also be regarded as the study of fault tolerance in networks with faults. Parallel routing on networks without faulty nodes has been studied, but there does not appear to be a systematic study on parallel routing on networks with faults. We study this strong fault tolerance of two extensively studied interconnection structures, the star networks and the hypercube networks. First, we give a brief review of previous related research on these networks.

The n-dimensional star network S_n is an undirected graph consisting of nodes of degree $n-1$. The star network has received considerable attention as an attractive alternative to the hypercube network model because of its rich structure, smaller diameter, lower degree, and symmetry properties [2, 54].

Parallel routing on star networks without faulty nodes has been studied in literature. Sur and Srimani [54] demonstrated that $n-1$ node-disjoint paths can be constructed between any two nodes in S_n in polynomial time. Dietzfelbinger, Madhavapeddy, and Sudborough [18] derived an improved algorithm that constructs $n-1$ node-disjoint paths of length bounded by 4 plus the diameter of S_n. The algorithm was further improved by Day and Tripathi [17] who developed an efficient algorithm that constructs $n-1$ node-disjoint paths of length bounded by 4 plus the shortest path length between any two nodes in the star networks. In particular, Chen and Chen [7] developed an efficient algorithm that constructs $n-1$ node-disjoint paths of optimal length for any two given nodes in S_n. Chen and Chen [8] also studied the problem of constructing node-disjoint paths between a node and a set of nodes. Gu and Peng [26] studied the problem of constructing node-disjoint paths connecting a set of nodes in star networks.

Fault tolerance on the star networks has been studied. The general fault tolerance properties of the star networks were first studied and analyzed in [1, 2, 3]. The problem of determining the diameter of a star network with faults was considered in [3, 23, 34, 51]. Algorithms for node-to-node routing in star networks with faults were developed in [6, 25]. Broadcasting algorithms in star networks with faults have been considered by a number of researchers [6, 21, 23, 36, 42].

The n-dimensional hypercube network Q_n is an undirected graph consisting of nodes of degree n. The hypercube networks are among the earliest and remain as one of the most important and attractive network models for multicomputer systems. It has been used for designing various commercial multiprocessor machines [53].

Saad and Schultz [52] first studied parallel routing on hypercube networks without faulty nodes. Madhavapeddy and Sudborough [40] developed an algorithm that constructs node-disjoint paths between disjoint source-destination pairs in the hypercube networks. Gu and Peng [28] also proposed an efficient algorithm for the pairwise disjoint paths between disjoint source-destination pairs. Latifi, Ko, and Srimani [35] provided a simple algorithm that constructs disjoint paths between one node and a set of nodes. Krishnamoorthy and Krishnamurthy [32] considered the problem of determining the diameter of hypercube networks with faulty nodes. Under the constraint that each non-faulty node must have at least one non-faulty neighbor, Latifi [33] studied the node-disjoint paths in hypercube networks with faulty nodes. Latifi used the node-disjoint paths as the method to derive the diameter of the hypercube network with faulty nodes. Many fault-tolerant communication algorithms concentrating on one-to-one routing or broadcasting in hypercube networks have been proposed [12, 13, 20, 24, 37, 38, 44, 50].

There are studies on fault tolerant routings involving the *surviving route graph*, where the diameter of the surviving graph is a measure of the worst case time to complete a broadcast. Dolev, Halpern, Simons, and Strong [19] studied the effects of faulty nodes and edges on the diameter of the surviving route graph. Broder, Dolev, Fischer, and Simons [5] applied the concept in [19] to the product graph, a cartesian product of component graphs. They derived the fault tolerant properties of the product graphs from the analysis of the surviving route graphs on the component graphs. Peleg and Simons [47] further studied a group of graphs such that the diameter of the surviving graph is bounded by a constant. They developed a routing method called the *kernel construction*, which can be used to construct disjoint paths between two given nodes. Based on the concept of surviving route graph, Rescigno and Vaccaro [49, 51] studied the fault tolerance of star and hypercube networks. In addition, Rescigno [49] applied the kernel construction approach to study randomized parallel routing in star networks. Rescigno's algorithm is randomized, thus, it does not always guarantee the maximum number of node-disjoint paths.

Also, there are related studies that utilize node or edge-disjoint paths on other fault-prone communication networks such as optical and mobile networks. In such networks, connectivity is related to the concept of network survivability that deals with a mechanism to protect resources against failures. A common approach to recover from failures is to provide an alternate path. In optical routing, link failures are common due to backhoe accidents. Choi, Subramaniam, and Choi [15, 16] utilized edge-disjoint paths to construct backup paths for failed edges. In mobile networks, multiple-path methods have been studied for designing routing algorithms to deliver messages with high success rates and low flooding rates. Lin and Stojmenovic [39] showed that desirable success rates and flooding rates can be achieved by using c disjoint paths, where c is a small constant. Other routing schemes that utilize multiple paths have been proposed [43, 45, 55]. In addition to study of disjoint paths, there are some efforts to characterize connectivity and fault tolerance of mobile networks in different contexts [22, 31, 46]. Specifically, Goyal and Caffery [22] suggest an approach that builds upon connectivity concepts in graph theory.

First, we study the strong fault tolerance for the star networks. Taking advantage of the *orthogonal decomposition* of the star networks, we develop an efficient algorithm that constructs node-disjoint paths between any two non-faulty nodes in n-star network S_n with at most $n-3$ faulty nodes: for any two non-faulty nodes u and v, our algorithm constructs $\min\{deg_f(u), deg_f(v)\}$ node-disjoint paths of minimum length plus a small constant between u and v.

Hypercube networks do not have a similar orthogonal decomposition structure. Thus, the techniques in parallel routing for the star networks with faults are not applicable to hypercube networks. In order to effectively route parallel paths in the hypercube networks with faults, we develop new techniques that pre-match the neighbors of two given nodes. Based on these techniques, we develop an algorithm that constructs $\min\{deg_f(u), deg_f(v)\}$ node-disjoint paths of optimal length for any pair of nodes u and v in the n-dimensional hypercube network with at most $n-2$ faulty nodes.

For both the star networks and the hypercube networks, the time complexity of our algorithm is optimal.

C. Routing in Hypercube Networks with High Success Probability

Since n-dimensional hypercube Q_n is a regular graph of degree n, the fault tolerance of Q_n is $n-1$. The fault tolerance $n-1$ reflects only a very rare situation. In most cases, the hypercube network is still connected when there are more than $n-1$ faulty nodes. Though nodes in the hypercube network are disconnected, there is a chance that a routing path can be constructed between two given nodes. Recently, Chen, Wang, and Chen [10, 11] introduced a new model of connectivity where local connectivity of small subcubes implies the global connectivity of the whole hypercube network. The advantage of using their model is that a probability derived for a subpath in a subcube or between subcubes allows us to derive the probability for the path in the entire hypercube network. Based on their model, we find a routing path consisting of subpaths between subcubes plus subpaths inside subcubes. If, with very high probability, we can construct a subpath between two subcubes or inside a subcube, then the probability that we can find a path from the source node to the destination node will be high. In this research, we find an algorithm that, with very high success probability, can construct a fault-free path without considering the global connectivity of the hypercube network.

Substantial work has been done on fault tolerant routing in hypercube networks. For example, Gu and Peng [24, 27] provide a fault tolerant routing scheme. We introduce research on fault tolerance in hypercube networks with a probabilistic approach. Chen and Shin [14] proposed a routing scheme using depth-first search with an arbitrary number of faulty nodes. They showed that depth-first search routing can use an optimal path with very high probability. Chen, Wang, and Chen [11] analyzed the probability of the global connectivity of a hypercube network with faulty nodes. They partitioned a hypercube network into subcubes, and derived the probability of the global connectivity of the entire hypercube network from the local connectivity of each subcube. Chen, Kanj, and Wang [9] derived lower bounds for the probability of fault tolerance of hypercube networks with a large number of faulty nodes. Their probability analysis is based on the measure for connectivity of hypercube networks.

D. Scope and Organization of the Dissertation

The objective of this research is to further study parallel routing and fault tolerance on contemporary interconnection networks. The dissertation is organized as follows:

In Chapter II, we study parallel routing in star networks with faulty nodes. First, we present the concept of *bridging paths* that connect a given node to a specific substar network in the star network. Based on this concept, we develop an efficient algorithm that constructs node-disjoint paths between any two non-faulty nodes in the n-star network with at most $n - 3$ faulty nodes: for any two non-faulty nodes u and v in the star network, our algorithm to find $\min\{deg_f(u), deg_f(v)\}$ node-disjoint paths. In Chapter III, we continue studying parallel routing in hypercube networks with faulty nodes. The techniques used in the study of star networks are not applicable to hypercube networks. We provide new approaches that construct node-disjoint paths between pairs of neighbors of two given nodes. Our study is divided into two cases depending on whether neighbors of two given nodes are faulty or not. In case there are no faulty neighbors for both source and destination nodes, we pre-pair the neighbors of the source and the destination nodes by a process, called **Prematch-I**. There is a special situation that may block all possible sets of parallel paths between two neighbors of the source and the destination nodes induced from **Prematch-I**. In this situation, we use a different process, called **Prematch-II**. The third process **Prematch-III** covers the case where there is at least one faulty neighbor of the source or the destination. In Chapter IV, we discuss routing algorithms on hypercube networks that lead to different success probabilities. The success probability of each routing algorithm is analyzed as well as time complexity and the length of path. Numerical results on success probabilities for each routing algorithm are given for hypercube networks whose dimension is selected between 10 and 40. In Chapter V, we conclude the dissertation, in which the major contributions of this dissertation are summarized, along with future research directions.

CHAPTER II

STRONG FAULT TOLERANCE OF THE STAR NETWORKS

A. Chapter Overview

In this chapter, we study the strong fault tolerance for the star networks. The n-dimensional star network S_n (or simply the n-star network) is an undirected graph consisting of $n!$ nodes labeled with $n!$ permutations on symbols $\{1, 2, \ldots, n\}$. There is an edge between two nodes u and v in S_n if and only if the permutation label for v can be obtained from the permutation label for u by exchanging the positions of the first symbol and another symbol, or by exchanging the first symbol and another symbol in u. Thus, the n-star graph has all its nodes of degree $n-1$. From the definition of strong fault tolerance, the number of faulty nodes that S_n can contain is less than $n-3$.

For the n-star network S_n, let $S_n[i]$ be the set of nodes in which the symbol 1 is at the ith position. Akers, Harel, and Krishnamurthy [1] showed that S_n can be decomposed into subsets such that the set $S_n[1]$ is an independent set, and the set $S_n[i]$ for $i \neq 1$ is an $(n-1)$-star network. This decomposition is called *orthogonal decomposition structure* of the star networks. We observe that the orthogonal partition of star networks seems very convenient for the construction of node-disjoint paths. We basically can construct a path in each substar which ensures that the constructed path in each substar is node-disjoint from the other paths. Since we assume that an n-star network can have at most $n-3$ faulty nodes, a fault-free path in each $(n-1)$-substar can be obtained by applying Day and Tripathi's algorithm [17]. From this observation, we developed the concept of *bridging paths* that connect a given node to a specific substar network in the n-star network. We develop our parallel routing algorithm based heavily on this concept of bridging paths.

We show that the star networks are strongly fault tolerant and develop an efficient algorithm that constructs node-disjoint paths between any two nodes in the n-star network S_n with at most $n-3$ faulty nodes: for any two non-faulty nodes u and v in the network, our algorithm constructs, in time $O(n^2)$, $\min\{deg_f(u), deg_f(v)\}$ node-disjoint paths of minimum length plus a small constant between u and v. The time complexity of our algorithm is optimal and our algorithm requires no prior knowledge of faulty nodes.

B. Properties of Star Networks

A permutation $u = \langle a_1 a_2 \cdots a_n \rangle$ of the symbols $1, 2, \ldots, n$ can be given by a product of disjoint cycles [4], which is called the *cycle structure* of the permutation. A cycle is *nontrivial* if it contains more than one symbol. Otherwise the cycle is *trivial.* The cycle containing the symbol 1 will be called the *primary cycle.* For example, the permutation $\langle 32541 \rangle$ has the cycle structure $(351)(2)(4)$. The cycles can be interpreted as follows: the primary cycle (351) indicates that 3 is at 1's position, 5 is at 3's position, and 1 is at 5's position. The trivial cycles (2) and (4) indicate that 2 and 4 are in their "correct" positions.

We define two groups of operations ρ_i and σ_a on permutations as follows. Given a permutation u, for each position i, $1 \le i \le n$, and for each symbol a in $\{1, 2, \ldots, n\}$, $\rho_i(u)$ is the permutation obtained from u by exchanging the first symbol and ith symbol in u, and $\sigma_a(u)$ is the permutation obtained from u by exchanging the first symbol and the symbol a in u.

Let us consider how these operations change the cycle structure of a permutation.

Write u in its cycle structure

$$u = (a_{11} \cdots a_{1n_1}\ 1)(a_{21} \cdots a_{2n_2}) \cdots (a_{k1} \cdots a_{kn_k})$$

If the ith symbol a of u is not in the primary cycle, then $\rho_i(u) = \sigma_a(u)$ "merges" the cycle containing a into the primary cycle. More precisely, suppose that $a = a_{21}$ (note that each cycle can be cyclically permuted and the order of the cycles is irrelevant), then the permutation $\rho_i(u) = \sigma_a(u)$ will have the cycle structure:

$$\rho_i(u) = \sigma_a(u) = (a_{21} \cdots a_{2n_2} a_{11} \cdots a_{1n_1}\ 1)(a_{31} \cdots a_{3n_3}) \cdots (a_{k1} \cdots a_{kn_k})$$

If the ith symbol a is in the primary cycle, then $\rho_i(u) = \sigma_a(u)$ "splits" the primary cycle into two cycles. More precisely, suppose that $a = a_{1j}$, where $1 \leq j \leq n_1+1$ (here we have let $a_{1n_1+1} = 1$), then $\rho_i(u) = \sigma_a(u)$ will have the following cycle structure:

$$\rho_i(u) = \sigma_a(u) = (a_{11} \cdots a_{1j-1})(a_{1j} \cdots a_{1n_1} 1)(a_{21} \cdots a_{2n_2}) \cdots (a_{k1} \cdots a_{kn_k})$$

In particular, if $a = a_{12}$, then we say that the operation ρ_i "deletes" the symbol a_{11} from the primary cycle.

Since the nodes in the n-star network S_n are labeled by the permutations on the symbols $\{1, 2, \ldots, n\}$, throughout this paper, we assume that each node in the n-star network S_n is given by its corresponding permutation. By the definition of the n-star network S_n, each node u is adjacent to the $n-1$ nodes $\rho_i(u)$, $2 \leq i \leq n$. Equivalently, the $n-1$ neighbors of u are the $n-1$ permutations $\sigma_a(u)$, where a is any symbol in $\{1, 2, \ldots, n\}$ except the first symbol in u. A path in S_n from a node u to a node v corresponds to a sequence of nodes obtained by applying the operations ρ_i or σ_a, starting from the node u and ending at the node v.

Denote by ε the node labeled by the identity permutation, $\varepsilon = \langle 12 \cdots n \rangle$. Since the n-star network S_n is vertex-symmetric [2], a set of node-disjoint paths from a node u to a node v can be mapped to a set of node-disjoint paths from a node u' to ε in a straightforward way. Therefore, we will concentrate on the construction of node-disjoint paths from u to ε in S_n.

Denote the distance from a node u to ε by $dist(u)$. Let u have the cycle structure $u = c_1 \cdots c_k e_1 \cdots e_m$, where c_i are nontrivial cycles and e_j are trivial cycles. If we further let $l = \sum_{i=1}^{k} |c_i|$, where $|c_i|$ denotes the number of symbols in the cycle c_i, then the distance $dist(u)$ from the node u to the identity node ε is given by the following formula [2].

$$dist(u) = \begin{cases} l + k & \text{if the primary cycle is a trivial cycle} \\ l + k - 2 & \text{if the primary cycle is a nontrivial cycle} \end{cases}$$

Combining this formula with the above discussion on the effect of applying the operations ρ_i and σ_a on a permutation, we derive the following necessary and sufficient rules for tracing a shortest path from the node u to the identity node ε in the n-star network S_n.

Shortest Path Rules

Rule 1. If the primary cycle is a trivial cycle in u, then in the next node on any shortest path from u to ε, a nontrivial cycle c_i is merged into the primary cycle. This corresponds to applying the operation σ_a on u with $a \in c_i$;

Rule 2. If the primary cycle $c_1 = (a_{11} a_{12} \cdots a_{1n_1} a_{1n_1+1})$ is a nontrivial cycle in u, where $a_{1n_1+1} = 1$, then in the next node on any shortest path from u to ε, either a nontrivial cycle $c_i \neq c_1$ is merged into the primary cycle (this corresponds to applying the operation σ_a on u, where $a \in c_i$), or the symbol a_{11} is deleted from the primary cycle c_1 (this corresponds to applying the operation $\sigma_{a_{12}}$ on u).

Fact 2.1. A shortest path from u to ε in S_n is obtained by a sequence of applications of the Shortest Path Rules, starting from the permutation u.

Fact 2.2. If an edge $[u, v]$ in S_n does not lead to a shortest path from u to ε, then $dist(v) = dist(u) + 1$. Consequently, let P be a path from u to ε in which exactly k edges do not follow the Shortest Path Rules, then the length of the path P is equal to $dist(u) + 2k$.

Two simple procedures will be used in following a shortest path from a node u to ε. The first is called the "Delete" procedure, written as $\rightarrow \overset{D}{\cdots} \rightarrow$, which repeatedly deletes the first symbol in the non-trivial primary cycle. The second one is called the "Merge-Delete" procedure [7], written as $\rightarrow \overset{M+D}{\cdots} \rightarrow$, which works in two stages: first repeatedly merges in an arbitrary order each of the nontrivial cycles into the primary cycle, then repeatedly deletes the first symbol in the primary cycle. It is easy to verify that both the "Delete" procedure and the "Merge-Delete" procedure follow the Shortest Path Rules strictly.

For the n-star network S_n, let $S_n[i]$ be the set of nodes in which the symbol 1 is at the ith position. Then the set $S_n[1]$ is an independent set (i.e., no two nodes in $S_n[1]$ are adjacent to each other), and the subgraph induced by the set $S_n[i]$ for $i \neq 1$ is a $(n-1)$-dimensional star network (which will also be called a "substar" and denoted as $S_n[i]$ without any confusion). Note that a node is in the substar $S_n[i]$, $i \neq 1$, if and only if the primary cycle of the node is of form $(\cdots i1)$, and a node is in $S_n[1]$ if and only if the primary cycle of the node is a trivial cycle (1).

A nice property of the Delete procedure and Merge-Delete procedure is that if they start with a node u in the substar $S_n[i]$, $i \neq 1$, then all nodes, possibly except the last one, on the constructed shortest path are also in the substar $S_n[i]$.

C. Bridging Paths from a Node to a Substar

Our parallel routing algorithm is heavily based on the concept of *bridging paths* that connect a given node to a specific substar network in the n-star network. In this section, we give formal definitions for bridging paths, and study its properties. We will also consider the complexity of extending a bridging path into a path from the given node u to the node ε.

The following lemma will serve as a basic tool in our construction of node-disjoint paths in the parallel routing algorithm.

Lemma C.1 *Let u be any non-faulty node in the substar $S_n[i]$ with $k_i \leq n-3$ faulty nodes, $i \neq 1$. A fault-free path P from u to $\rho_i(\varepsilon)$ can be constructed in $S_n[i]$ in time $O(k_i n + n)$ such that at most two edges in P do not follow the Shortest Path Rules. In case the primary cycle of u is $(i1)$, the constructed path P has at most one edge not following the Shortest Path Rules.*

PROOF. We first assume that the substar $S_n[i]$ has no faulty nodes and show that there are $n-2$ node-disjoint paths from u to $\rho_i(\varepsilon)$ in $S_n[i]$.

The node u in $S_n[i]$ has the cycle structure of form $(a_1 a_2 \cdots a_p i1) * **$, where "$* * *$" stands for the "other cycles" in u.

The $n-2$ node-disjoint paths in $S_n[i]$ from the node u to the node $\rho_i(\varepsilon)$ are constructed as follows (note that the node $\rho_i(\varepsilon)$ has only one nontrivial cycle that is of form $(i1)$), where the sequences $\rightarrow \stackrel{D}{\cdots} \rightarrow$ and $\rightarrow \stackrel{M+D}{\cdots} \rightarrow$ are the "Delete" and the "Merge-Delete" procedures described in Section B.

For each h, $1 \leq h \leq p$, we construct a path from u to $\rho_i(\varepsilon)$ in $S_n[i]$, as follows:

$$u = (a_1 \cdots a_p i1) * ** \rightarrow (a_{h+1} \cdots a_p i1)(a_1 \cdots a_h) * ** \rightarrow \stackrel{D}{\cdots} \rightarrow \qquad (2.1)$$

$$(i1)(a_1 \cdots a_h) * ** \to (a_1 \cdots a_h i1) * ** \to \overset{M+D}{\cdots} \to (a_h i1) \to (i1) = \rho_i(\varepsilon) \tag{2.2}$$

This path is disjoint with the other constructed paths because the first part of it contains a unique cycle $(a_1 \cdots a_h)$ while the second part of it contains a cycle with a unique pattern $(\cdots a_h i1)$. It is easy to verify that at most two edges in this path do not follow the Shortest Path Rules.

For each symbol b not in the primary cycle, we construct a path from u to $\rho_i(\varepsilon)$ in $S_n[i]$ as follows (note that the symbols in each cycle can be cyclically rotated):

$$u = (a_1 \cdots a_p i1) * ** \xrightarrow{\sigma_b} (b \cdots b' a_1 \cdots a_p i1) * ** \to \overset{D}{\cdots} \to \tag{2.3}$$

$$\to (b' a_1 \cdots a_p i1) * ** \to (b' a_1 \cdots a_p)(i1) * ** \to (a_1 \cdots a_p b' i1) * ** \to \tag{2.4}$$

$$\to \overset{D}{\cdots} \to (b' i1) * ** \to \overset{M+D}{\cdots} \to (b' i1) \to (i1) \tag{2.5}$$

This path is disjoint with the other constructed paths because the first part of it contains a cycle of a unique pattern $(\cdots b' a_1 \cdots a_p i1)$ (note that each different symbol b gives a different symbol b'), and the second part of it contains a cycle of a unique pattern $(\cdots b' i1)$. Note that if b is in a nontrivial cycle in u, then the first edge in the path follows the Shortest Path Rules, and the path has at most one edge not following the Shortest Path Rules. On the other hand, if b is in a trivial cycle in u, then the first edge does not follow the Shortest Path Rules, which gives $(b a_1 \cdots a_p i1) * **$, so we directly get the first pattern in line (2.4), and the rest of the part of the path has at most one edge not following the Shortest Path Rules (note that if $p = 0$ then the node $(b' a_1 \cdots a_p i1) * **$ is the same as the node $(b' i1) * **$ in line (2.5)). Thus, in any case, the path has at most two edges not following the Shortest Path Rules.

Note that lines (2.1)-(2.5) construct exactly $n - 2$ node-disjoint paths in $S_n[i]$. It is also easy to verify that the construction of each of these paths takes time $O(n)$, and is independent of the construction of the other paths.

In case the node u has a primary cycle $(i1)$, all the $n-2$ paths are constructed based on the lines (2.4)–(2.5). Let b be any symbol, $b \neq 1, i$. In this case, if b is in a trivial cycle in u, then the second node on the path is $(bi1) * **$ (the edge from u to this node does not follow the Shortest Path Rules). Thus, the sequence $\rightarrow \overset{M+D}{\cdots} \rightarrow$ in line (2.5) can be applied to $(bi1) * **$ directly in which all edges follow the Shortest Path Rules. Thus, there is exactly one edge on the path not following the Shortest Path Rules. On the other hand, if b is in a nontrivial cycle in u, then the first edge from u to $(b \cdots b'i1) * **$ follows the Shortest Path Rules. Thus, after the sequence $\rightarrow \overset{D}{\cdots} \rightarrow$ in line (2.4), we arrive at a node of form $(b'i1)$, on which again the sequence $\rightarrow \overset{M+D}{\cdots} \rightarrow$ in line (2.5) can be applied. Thus, in this case, the constructed path is actually a shortest path.

This concludes that in case the primary cycle of u is $(i1)$, each of the $n-2$ node-disjoint paths constructed in (2.1)-(2.5) contains at most one edge not following the Shortest Path Rules.

Since there are $k_i \leq n-3$ faulty nodes in the substar $S_n[i]$, at least one of the above $n-2$ constructed paths contains no faulty nodes. Such a path can be found by tracing at most $k_i + 1$ of the above $n-2$ node-disjoint paths. Since tracing each such path takes time $O(n)$ and is independent of the construction of the other paths, we conclude that a fault-free path P from u to $\rho_i(\varepsilon)$ in the substar $S_n[i]$ can be constructed in time $O(k_i n + n)$ such that the path P contains at most two edges not following the Shortest Path Rules, and in case the primary cycle of u is $(i1)$, the path P contains at most one edge not following the Shortest Path Rules. $\square$

The construction of the $n-2$ node-disjoint paths from u to $\rho_i(\varepsilon)$ in the substar $S_n[i]$ in the proof of Lemma C.1 is essentially a modification of the parallel routing algorithm developed in [17], with the n-star network being replaced by the substar

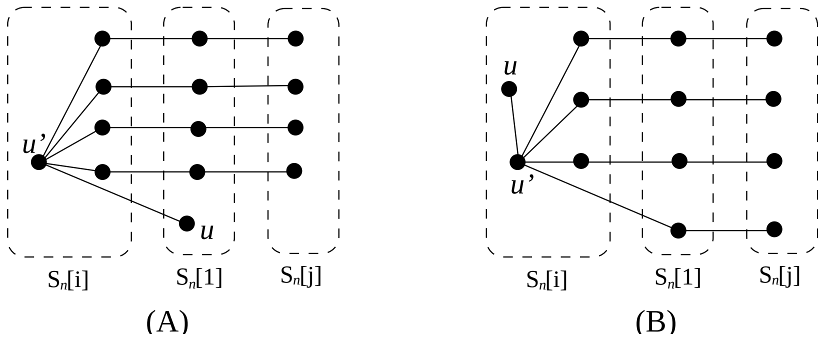

Fig. 1. Bridging paths from node u to substar $S_n[j]$: (A) u is in $S_n[1]$; (B) u is in $S_n[i]$

$S_n[i]$.

Let u be a node in the n-star network S_n, and let u' be a neighbor of u in the substar $S_n[i]$, where $i \neq 1$. From each neighbor v of u', we can construct a path from u, via u' and v, to the substar $S_n[1]$ then to a substar $S_n[j]$, where $j \neq 1, i$, as follows: if v is in $S_n[1]$, then the next node is $\rho_j(v)$, while if v is in $S_n[i]$, then the next two nodes are $v' = \rho_i(v)$ and $v'' = \rho_j(v')$. This structure will be very important in our construction of node-disjoint paths in the n-star network. Let us formally define this as follows:

Definition Let u be a node in the n-star network S_n and u' be a neighbor of u in the substar $S_n[i]$, $i \neq 1$. For each neighbor v of u', $v \neq u$, a (u', j)-*bridging path* (of length at most 4) from u to the substar $S_n[j]$, $j \neq 1, i$, is defined as follows: if v is in $S_n[1]$ then the path is $[u, u', v, \rho_j(v)]$, while if v is in $S_n[i]$ then the path is $[u, u', v, \rho_i(v), \rho_j(\rho_i(v))]$.

Thus, from each neighbor u' in $S_n[i]$ of the node u, $i \neq 1$, there are $n-2$ (u', j)-bridging paths of length bounded by 4 that connect the node u to the substar $S_n[j]$. See Fig. 1 for an intuitive illustration for bridging paths.

Since no two nodes in $S_n[i]$ share the same neighbor in $S_n[1]$ and no two nodes in $S_n[1]$ share the same neighbor in $S_n[j]$, for any neighbor u' of u, two (u', j)-bridging paths from u to $S_n[j]$ have only the nodes u and u' in common. Moreover, for any two neighbors u' and u'' of u in $S_n[i]$ (in this case, the node u must itself also be in $S_n[i]$), since u' and u'' have no other common neighbor except u (see, for example, [8, 17]), a (u', j)-bridging path from u to $S_n[j]$ and a (u'', j)-bridging path from u to $S_n[j]$ share no nodes except u.

Definition Let u be a node in S_n and let u' be a neighbor of u in $S_n[i]$, $i \neq 1$. A (u', j)-bridging path P from the node u to the substar $S_n[j]$ is *divergent* if in the subpath of P from u to $S_n[1]$, there are three edges not following the Shortest Path Rules.

Note that the subpath from u to $S_n[1]$ of a (u', j)-bridging path P contains at most three edges. In particular, if the subpath contains only two edges, then the path P is automatically non-divergent.

In case there are no faulty nodes in the n-star network, each divergent (u', j)-bridging path can be efficiently extended into a path from u to $\rho_j(\varepsilon)$, as shown in the following lemma.

Lemma C.2 *There is an $O(n)$ time algorithm that, given a divergent (u', j)-bridging path P from a node u to a substar $S_n[j]$, extends P into a path Q from u to $\rho_j(\varepsilon)$, such that at most 4 edges in Q do not follow the Shortest Path Rules, and the extended part is entirely in the substar $S_n[j]$. Moreover, for two divergent (u', j)-bridging paths P_1 and P_2, the two corresponding extended paths Q_1 and Q_2 have only the nodes u, u', and $\rho_j(\varepsilon)$ in common.*

Proof. Let P be a divergent (u', j)-bridging path from the node u to the substar $S_n[j]$, where u' is a neighbor of u. Since the path P is divergent, it has length 4 and the node u' is in $S_n[i]$, $i \neq 1$. Thus, the path P can be written as $P = \{u, u', v, v', v''\}$, where u' is in $S_n[i]$, v is a neighbor of u' in $S_n[i]$, $v' = \rho_i(v)$ is in $S_n[1]$, and $v'' = \rho_j(v')$ is in $S_n[j]$.

Let $u' = (a_1 \cdots a_p i1) * **$, where "$* * *$" stands for "other cycles". Since the edge $[u', v]$ does not follow the Shortest Path Rules and v is in $S_n[i]$, the node v must have the form either $v = (ba_1 \cdots a_p i1) * **$, where (b) is a trivial cycle in u', or $v = (a_1 \cdots a_q)(a_{q+1} \cdots a_p i1) * **$, where $2 \leq q \leq p$. Now, since $[v, v']$ is an edge in S_n and v' is in $S_n[1]$, the node v' must be of the form either $v' = (ba_1 \cdots a_p i)(1) * **$, or $v' = (a_{q+1} \cdots a_p i)(1) * **$. Moreover, since the edge $[v, v']$ does not follow the Shortest Path Rules, when $v' = (a_{q+1} \cdots a_p i)(1) * **$, we must have $q + 1 \leq p$. In summary, if P is a divergent path, then the fourth node v' on P must be of form $(b_1 b_2 \cdots i)(1)$, where the cycle $(b_1 b_2 \cdots i)$ is non-trivial. Moreover, the (u', j)-bridging path P is distinguished from other (u', j)-bridging paths by the symbol b_1 in the above format (i.e., two different divergent (u', j)-bridging paths will have two different symbols b_1 in the above format).

Now consider the fourth edge $[v', v'']$ on the path P, where v'' is in $S_n[j]$, $j \neq 1, i$. If the symbol j is in a trivial cycle in the node v', then j is not in the non-trivial cycle $(b_1 b_2 \cdots i)$. The extended path Q is obtained by:

$$
\begin{aligned}
Q:\quad & u \to u' \to v \to v' = (b_1 b_2 \cdots i)(1) * ** \to v'' = (b_1 b_2 \cdots i)(j1) * ** \to \\
& \to (b_2 \cdots i b_1 j1) * ** \to \overset{M+D}{\cdots} \to (b_1 j1) \to (j1) = \rho_j(\varepsilon) \qquad (2.6)
\end{aligned}
$$

The extended path Q has no common nodes in $S_n[j]$, except $\rho_j(\varepsilon)$, with the paths extended from the other (u', j)-bridging paths since the symbol b_1 distinguishes the path Q from other extended paths: the first part of Q has a unique cycle $(b_1 b_2 \cdots i)$ while the second part of Q has a cycle of the unique format $(\cdots b_1 j 1)$.

If the symbol j is in a non-trivial cycle in the node v', then there are two possible cases:

Case 1. The symbol j is not in the cycle $(b_1 b_2 \cdots i)$. The extended path Q is:

$$\begin{aligned} Q: \quad & u \to u' \to v \to v' = (b_1 b_2 \cdots i)(1) * ** \to v'' = (b_1 b_2 \cdots i)(\cdots j1) * ** \to \\ & \to \overset{D}{\cdots} \to (b_1 b_2 \cdots i)(j1) * ** \to (b_2 \cdots i b_1 j 1) * ** \to \overset{M+D}{\cdots} \to \\ & \to (b_1 j 1) \to (j1) = \rho_j(\varepsilon) \end{aligned} \tag{2.7}$$

Again, because of the symbol b_1, the extended path Q has no common nodes in $S_n[j]$, except $\rho_j(\varepsilon)$, with the paths extended from the other (u', j)-bridging paths.

Case 2. The symbol j is in the cycle $(b_1 b_2 \cdots i)$.

If $j = b_1$, then $(b_1 b_2 \cdots i) = (b_2 \cdots i j)$, and the path Q is:

$$\begin{aligned} Q: \quad & u \to u' \to v \to v' = (b_2 \cdots i j)(1) * ** \to v'' = (b_2 \cdots i j 1) * ** \to \\ & \to \overset{M+D}{\cdots} \to (ij1) \to (j1) = \rho_j(\varepsilon) \end{aligned} \tag{2.8}$$

This path is node-disjoint from the paths extended from the other (u', j)-bridging paths because all nodes of it in $S_n[j]$ contain a cycle with a unique format $(\cdots ij1)$.

If $j \neq b_1$, then $(b_1 b_2 \cdots i) = (b_1 \cdots j \cdots i)$, and the path Q is:

$$\begin{aligned} Q: \quad & u \to u' \to v \to v' = (b_1 \cdots j \cdots i)(1) * ** \to v'' = (\cdots i b_1 \cdots j 1) * ** \to \\ & \to (\cdots i b_1 \cdots)(j1) * ** \to (\cdots i b_1 j 1) \overset{M+D}{\cdots} \to (b_1 j 1) \to (j1) = \rho_j(\varepsilon) \end{aligned} \tag{2.9}$$

Again this path is node-disjoint from the paths extended from the the other (u', j)-bridging paths because of the symbol b_1.

For all cases, we can easily verify that the constructed path Q contains at most 4 edges not following the Shortest Path Rules and that the part of Q extended from the (u', j)-bridging path P is entirely in the substar $S_n[j]$. Finally, from the sequences (2.6)-(2.9), it can be easily seen that the construction of the extended path Q takes time $O(n)$ and is independent of the construction of other extended paths. $\square$

D. Parallel Routing Algorithm on Faulty Star Networks

We present our parallel routing algorithm on star networks with faults in Fig. 2.

Suppose that the n-star network S_n has at most $n-3$ faulty nodes. Assume that the node ε is non-faulty. For each non-faulty node u in S_n, let $deg_f(u)$ be the degree of the node u in S_n with the faulty nodes removed. For any given non-faulty node u in the star network with faults, our algorithm constructs $\min\{deg_f(u), deg_f(\varepsilon)\}$ node-disjoint fault-free paths from u to ε such that the length of the paths is bounded by $dist(u) + 8$. We provide more detailed explanations for each step of the algorithm below.

Step 1 of the algorithm constructs certain number of paths between non-faulty neighbors of the node u and non-faulty neighbors of the node ε. Step 2 of the algorithm maximally pairs the rest of the non-faulty neighbors of u with the rest of the non-faulty neighbors of ε. It is easy to see that the number g of pairs constructed in Step 2 plus the number of paths constructed in Step 1 is exactly $\min\{deg_f(u), deg_f(\varepsilon)\}$. Since Step 3 of the algorithm constructs a path from u to ε for each pair constructed in Step 2, the algorithm **Parallel-Routing-Star** constructs exactly $\min\{deg_f(u), deg_f(\varepsilon)\}$ paths from u to ε. What remains to be seen is how these paths are constructed, in time $O(n^2)$, so that they are fault-free, node-disjoint, and of length bounded by $dist(u) + 8$.

Algorithm. Parallel-Routing-Star

Input: a non-faulty node u in the n-star network S_n with at most $n-3$ faulty nodes.

Output: $\min\{deg_f(u), deg_f(\varepsilon)\}$ node-disjoint fault-free paths of length $\leq dist(u)+8$ from u to ε.

1. **if** the node u is in $S_n[1]$
1.1. **then**
  for each index $j \neq 1$ such that both $\rho_j(u)$ and $\rho_j(\varepsilon)$ are non-faulty
  do
    construct a path P_j of length $\leq dist(u)+6$ from u to ε such that all internal nodes of the path are in $S_n[j]$;
1.2. **else** (* the node u is in a substar $S_n[i]$, $i \neq 1$ *)
1.2.1. **if** the node $\rho_i(\varepsilon)$ is non-faulty
  then pick a non-faulty neighbor v of u and construct a path P_v of length $\leq dist(u)+4$ from u to ε such that all internal nodes of P_v are in $S_n[i]$ and P_v does not intersect a (u', j)-bridging path for any non-faulty neighbor $u' \neq v$ of u;
1.2.2. **if** the neighbor $u_1 = \rho_i(u)$ of u in $S_n[1]$ is non-faulty
  then find an index j, $j \neq 1, i$, such that both $\rho_j(u_1)$ and $\rho_j(\varepsilon)$ are non-faulty, and extend the path $[u, u_1, \rho_j(u_1)]$ to a path P_1 of length $\leq dist(u)+8$ from u to ε such that all nodes between $\rho_j(u_1)$ and $\rho_j(\varepsilon)$ are in $S_n[j]$;

2. let $u'_1, \ldots, u'_s$ and $\rho_{j_1}(\varepsilon), \ldots, \rho_{j_t}(\varepsilon)$ be the non-faulty neighbors of u and ε, respectively, not used in step 1, maximally pair them: $(u'_1, \rho_{j_1}(\varepsilon)), \ldots, (u'_g, \rho_{j_g}(\varepsilon))$, where $g = \min\{s, t\}$;

3. **for** each pair $(u', \rho_j(\varepsilon))$ constructed in step 2 **do**
3.1. **if** there is a non-divergent (u', j)-bridging path P with neither faulty nodes nor nodes used by other paths
  then pick this (u', j)-bridging path P
  else pick a divergent (u', j)-bridging path P with neither faulty nodes nor nodes used by other paths;
3.2. extend the (u', j)-bridging path P into a fault-free path $P_{u'}$ of length $\leq dist(u)+8$ from u to ε such that the extended part in $P_{u'}$ is entirely in $S_n[j]$;

Fig. 2. Parallel routing on the star network with faulty nodes

Step 1 of the algorithm

In case the node u is in $S_n[1]$, for each index $j \neq 1$ such that both $\rho_j(u)$ and $\rho_j(\varepsilon)$ are non-faulty, we construct a path P_j from u to ε such that all internal nodes of P_j are in the substar $S_n[j]$. By Lemma C.1, we can construct in time $O(k_j n + n)$ a path Q_j without faulty nodes from $\rho_j(u)$ to $\rho_j(\varepsilon)$ in the substar $S_n[j]$ such that at most two edges in Q_j do not follow the Shortest Path Rules, where $k_j \leq n-3$ is the number of faulty nodes in the substar $S_n[j]$. Thus, the concatenation of the edge $[u, \rho_j(u)]$, the path Q_j, and the edge $[\rho_j(\varepsilon), \varepsilon]$ gives the path P_j without faulty nodes from u to ε in which at most three edges do not follow the Shortest Path Rules (note that the edge $[\rho_j(\varepsilon), \varepsilon]$ always follows the Shortest Path Rules). By Fact 2.2, the length of the path P_j is bounded by $dist(u) + 6$. This path P_j is disjoint with other paths constructed in Step 1 because all internal nodes of P_j are in the substar $S_n[j]$ while no other constructed paths use any node in $S_n[j]$.

In case the node u is in a substar $S_n[i]$, $i \neq 1$, we construct at most two paths from u to ε in Step 1.

If the node $\rho_i(\varepsilon)$ is non-faulty, by Lemma C.1, we can construct in time $O(k_i n+n)$ a path Q_v without faulty nodes from u to $\rho_i(\varepsilon)$ in the substar $S_n[i]$ (where v is the second node on the path) such that at most two edges in Q_v do not follow the Shortest Path Rules, where $k_i \leq n-3$ is the number of faulty nodes in $S_n[i]$. This path Q_v plus the edge $[\rho_i(\varepsilon), \varepsilon]$ gives a path P_v of length bounded by $dist(u) + 4$ in which all internal nodes are in the substar $S_n[i]$. Moreover, we need to show that the path P_v can be constructed without intersecting with any (u', j)-bridging path from u for any non-faulty neighbor $u' \neq v$ of u and any j. Suppose that the path P_v intersects some (u', j)-bridging paths for non-faulty neighbors $u' \neq v$ of u. Let w be the last node on P_v that belongs to a (u', j)-bridging path $Q_{u'}$ for a non-faulty neighbor $u' \neq v$ of u and for some index j. Note that the neighbor u' of u is uniquely determined by the node w since for two different neighbors u' and u'' of u in $S_n[i]$, a (u', j')-bridging path and a (u'', j'')-bridging path have no common nodes except u. Therefore, we can use the path $P_{u'}$ instead of the path P_v, where $P_{u'}$ is the subpath of $Q_{u'}$ from u to w plus the subpath of P_v from w to ε. It is easy to verify that the length of the path $P_{u'}$ is not larger than the length of the path P_v, and that the path $P_{u'}$ does not intersect any (u'', j'')-bridging path from u for any non-faulty neighbor $u'' \neq u'$ of u and for any $j'' \neq 1, i$.

If the neighbor $u_1 = \rho_i(u)$ of u in $S_n[1]$ is non-faulty, consider the $n-2$ pairs $(\rho_j(u_1), \rho_j(\varepsilon))$ of neighbors of u_1 and ε, where $j \neq 1, i$. Since the n-star network S_n has at most $n-3$ faulty nodes, one of these pairs $(\rho_j(u_1), \rho_j(\varepsilon))$ has both nodes non-faulty. By Lemma C.1, a fault-free path Q_1 from $\rho_j(u_1)$ to $\rho_j(\varepsilon)$ can be constructed in the substar $S_n[j]$ in time $O(k_j n + n)$ such that at most two edges of Q_1 do not follow the Shortest Path Rules, where $k_j \leq n-3$ is the number of faulty nodes in the substar $S_n[j]$. Now the concatenation P_1 of the path $[u, u_1, \rho_j(u_1)]$, the path Q_1, and the edge $[\rho_j(\varepsilon), \varepsilon]$ gives a fault-free path from u to ε of length bounded by $dist(u)+8$. Note that this path is obviously node-disjoint with the path constructed in Step 1.2.1.

Step 2 of the algorithm

It is easy to see that Step 2 of the algorithm takes time $O(n)$.

Step 3 of the algorithm

We consider Step 3 of the algorithm for two different cases.

Case 3.1. The node u is in $S_n[1]$.

Consider each pair $(\rho_h(u), \rho_j(\varepsilon))$ constructed in Step 2. Note we must have $h \neq j$, and the nodes $\rho_h(\varepsilon)$ and $\rho_j(u)$ must be faulty since otherwise the index h or the index j would have been picked in Step 1.1.

We construct a path Q_{hj} from u to ε by concatenating a $(\rho_h(u), j)$-bridging path from u to $S_n[j]$ with a path Q'_j entirely in the substar $S_n[j]$. Note that such a path Q_{hj} contains one node in $S_n[1]$ and all other nodes in $S_n[h]$ and $S_n[j]$. We say that a node in $S_n[1]$ is *occupied* if it has been used by a path Q_{hj} for a pair $(\rho_h(u), \rho_j(\varepsilon))$ constructed in Step 3 of the algorithm. Inductively, assume that for r pairs in Step 3 of the algorithm, r such node-disjoint paths satisfying the required conditions have been constructed, $r < g$. Now we consider the $(r+1)$st pair $(\rho_h(u), \rho_j(\varepsilon))$.

Each $(\rho_{h'}(u), \rho_{j'}(\varepsilon))$ of the previous r pairs implies at least two faulty nodes: the node $\rho_{h'}(\varepsilon)$ in $S_n[h']$ and the node $\rho_{j'}(u)$ in $S_n[j']$, and one occupied node in $S_n[1]$. Also notice that the paths constructed in Step 1.1 do not use any nodes in $S_n[1]$. Thus, the number of faulty nodes in the sets $S_n[1]$, $S_n[h]$, and $S_n[j]$ is at most $(n-3)-2r = n-2r-3$. Let k_j be the number of faulty nodes in $S_n[j]$, $k_j \leq n-2r-3$.

Case 3.1.A. There is a non-divergent $(\rho_h(u), j)$-bridging path $P_{hj} = [u, u', v, v', v'']$ with neither faulty nodes nor occupied nodes from u to $S_n[j]$. Thus, at least one of the first three edges of P_{hj} follows the Shortest Path Rules. Consider the last edge $[v', v'']$ on P_{hj}

If the edge $[v', v'']$ also follows the Shortest Path Rules, then the path P_{hj} has at most two edges not following the Shortest Path Rules. According to Lemma C.1, we can construct a path Q'_j without faulty nodes in the substar $S_n[j]$ from v'' to $\rho_j(\varepsilon)$ in time $O(k_j n + n)$ such that at most two edges in Q'_j do not follow the Shortest Path Rules. Now the concatenation of the $(\rho_h(u), j)$-bridging path P_{hj}, the path Q'_j, and the edge $[\rho_j(\varepsilon), \varepsilon]$ gives a path $P_{\rho_h(u)}$ without faulty nodes from u to ε such that at most 4 edges in $P_{\rho_h(u)}$ do not follow the Shortest Path Rules. By Fact 2.2, the length of the path $P_{\rho_h(u)}$ is bounded by $dist(u) + 8$.

If the edge $[v', v'']$ does not follow the Shortest Path Rules, then the path P_{hj} may have three edges not following the Shortest Path Rules. Since v' is in $S_n[1]$, it has the form $(1){*}{*}{*}$. Now v'' is in $S_n[j]$ and the edge $[v', v'']$ does not follow the Shortest Path

Rules. Thus, v'' must be of the form $(j1) * **$. By Lemma C.1, a path Q'_j without faulty nodes from v'' to $\rho_j(\varepsilon)$ in $S_n[j]$ can be constructed in time $O(k_j n + n)$ in which at most one edge does not follow the Shortest Path Rules. Now the concatenation of the $(\rho_h(u), j)$-bridging path P_{hj}, the path Q'_j, and the edge $[\rho_j(\varepsilon), \varepsilon]$ gives a path $P_{\rho_h(u)}$ without faulty nodes from u to ε in which at most 4 edges do not follow the Shortest Path Rules. By Fact 2.2, the path $P_{\rho_h(u)}$ has length of at most $dist(u) + 8$.

Therefore, in this case, for the pair $(\rho_h(u), \rho_j(\varepsilon))$ in Step 3 of the algorithm, we can always construct, in time $O(k_j n + n)$, a path $P_{\rho_h(u)}$ with neither faulty nodes nor occupied nodes and of length bounded by $dist(u) + 8$ from node u to node ε. This path is node-disjoint with all previously constructed paths since the part extended from the $(\rho_h(u), j)$-bridging path P_{hj} is entirely in the substar $S_n[j]$ that is not used by any other paths.

Case 3.1.B. All non-divergent $(\rho_h(u), j)$-bridging paths from u to $S_n[j]$ contain either faulty nodes or occupied nodes.

Total there are $n-2$ $(\rho_h(u), j)$-bridging paths from u to $S_n[j]$. Suppose that q' of them contain either faulty nodes or occupied nodes, and that $q = n - 2 - q'$ of them contain neither faulty nodes nor occupied nodes.

We first show $q > 0$. Assume the contrary $q = 0$. Then $q' = n-2$. Since any two $(\rho_h(u), j)$-bridging paths from u have only the nodes u and $\rho_h(u)$ in common and there are at most $n-3$ faulty nodes in S_n, q'_1 of these $n-2$ $(\rho_h(u), j)$-bridging paths contain only occupied nodes, where $q'_1 > 0$. Each of the rest $q'_2 = q' - q'_1 = n - q'_1 - 2$ $(\rho_h(u), j)$-bridging paths contains at least one faulty node. Thus, there are at least q'_1 occupied nodes. In consequence, at least q'_1 paths have been constructed by the algorithm for q'_1 pairs $(\rho_{h'}(u), \rho_{j'}(\varepsilon))$. (Note that each constructed path occupies exactly one node in the set $S_n[1]$.) Each $(\rho_{h'}(u), \rho_{j'}(\varepsilon))$ of these pairs implies two faulty nodes $\rho_{j'}(u)$ and $\rho_{h'}(\varepsilon)$, which cannot be on any of the $(\rho_h(u), j)$-bridging paths from u. Thus, the total number of faulty nodes in the n-star network S_n would have been at least $q'_2 + 2q'_1 = n + q'_1 - 2 > n - 3$, contradicting the assumption that the n-star network S_n has at most $n - 3$ faulty nodes. This shows $q > 0$, i.e., there is at least one $(\rho_h(u), j)$-bridging path that contains neither faulty nodes nor occupied nodes.

According to the assumption, the q $(\rho_h(u), j)$-bridging paths without faulty nodes and occupied nodes are all divergent. By Lemma C.2, these q $(\rho_h(u), j)$-bridging paths can be extended into q paths from u to $\rho_j(\varepsilon)$ such that each path contains at most 4 edges not following the Shortest Path Rules. The constructed paths contain no occupied nodes since the extended part of each path is entirely in the substar $S_n[j]$. Moreover, no two of these q paths share a node that is not u, $\rho_h(u)$, and $\rho_j(\varepsilon)$.

We claim that at least one of these q extended paths contains no faulty nodes. To the contrary, if each of these q extended paths contains at least one faulty nodes, then the total number of faulty nodes in the sets $S_n[1]$, $S_n[i]$, and $S_n[j]$ is at least $q + (q' - r) = n - r - 2 > n - 2r - 3$. (Recall that r is the number of paths that have been constructed by the algorithm so far. Thus, among the q' $(\rho_h(u), j)$-bridging paths that contain either faulty nodes or occupied nodes, at least $q' - r$ of them must contain at least one faulty node each.) This contradicts the fact that there are at most $n - 2r - 3$ faulty nodes in the sets $S_n[1]$, $S_n[i]$, and $S_n[j]$.

Thus, an extended path Q'_{hj} from u to $\rho_j(\varepsilon)$ with neither faulty nodes nor occupied nodes can be constructed. This path Q'_{hj} plus the edge $[\rho_j(\varepsilon), \varepsilon]$ gives a path $P_{\rho_h(u)}$ with neither faulty nodes nor occupied nodes from u to ε in which at most 4 edges do not follow the Shortest Path Rules. Thus, the length of the path $P_{\rho_h(u)}$ is bounded by $dist(u) + 8$. Moreover, the path $P_{\rho_h(u)}$ can be constructed in time $O(k_j n + n)$ by tracing at most $k_j + 1$ of the extended paths from u to $\rho_j(\varepsilon)$. Finally, this path is node-disjoint with all previously constructed paths since its extended part is entirely in the substar $S_n[j]$, which is not used by any other paths.

Case 3.2. The node u is in the substar $S_n[i]$, $i \neq 1$.

In this case, the node u has one neighbor in $S_n[1]$, and $n-2$ neighbors in $S_n[i]$ (see Figure 1). Note that if the neighbor $\rho_i(u)$ of u in $S_n[1]$ is non-faulty, then a path from u to ε via $\rho_i(u)$ has been constructed in Step 1.2.2. Thus, we only need to consider the neighbors of u that are in $S_n[i]$.

Again we assume that the algorithm has constructed r paths from u to ε by extending r bridging paths from u. Now consider the $(r+1)$st pair $(u', \rho_j(\varepsilon))$.

Since the n-star network contains no cycle of length less than 6 [8, 17], two neighbors of u share no common neighbors except u. Let u_1 and u_2 be two neighbors of u in $S_n[i]$. Since no two nodes in $S_n[i]$ have the same neighbor in $S_n[1]$ and no two nodes in $S_n[1]$ have the same neighbor in $S_n[j]$, a (u_1, j_1)-bridging path and a (u_2, j_2)-bridging path share no common nodes except u for any j_1 and j_2. Therefore, for the previous r paths from u to ε constructed by the algorithm by extending bridging paths from u, none of them would intersect a (u', j)-bridging path. Thus, no (u', j)-bridging path contains an occupied node.

Thus, if there is a non-divergent (u', j)-bridging path P_{hj} with no faulty nodes, we can extend the path P_{hj}, in the way of Case 3.1.A, into a path $P_{u'}$ from u to ε such that the length of the path $P_{u'}$ is bounded by $dist(u) + 8$, and the extended part of $P_{u'}$ is entirely in the substar $S_n[j]$. On the other hand, if all non-divergent (u', j)-bridging paths contain faulty nodes, then, as in Case 3.1.B, we can extend at least one divergent (u', j)-bridging path from u into a path $P_{u'}$ from u to ε such that the length of the path $P_{u'}$ is bounded by $dist(u) + 8$, and the extended part of $P_{u'}$ is entirely in the substar $S_n[j]$.

We are ready to state our main theorem.

Theorem D.1 *If the n-star network S_n has at most $n-3$ faulty nodes and the* *node ε is non-faulty, then for a non-faulty node u in S_n, in time $O(n^2)$ the algorithm* **Parallel-Routing-Star** *constructs* $\min\{deg_f(u), deg_f(\varepsilon)\}$ *node-disjoint fault-free paths of length bounded by* $dist(u) + 8$ *from the node u to the node ε.*

Proof. As we have discussed in detail above, the algorithm **Parallel-Routing-Star** constructs $\min\{deg_f(u), deg_f(\varepsilon)\}$ node-disjoint fault-free paths of length bounded by $dist(u) + 8$ from the node u to the node ε. The only thing remaining is to show that the running time of the algorithm is bounded by $O(n^2)$.

Each path is constructed by the algorithm by searching a proper path in a specific substar $S_n[j]$, which takes time $O(k_j n + n)$, where k_j is the number of faulty nodes in the substar $S_n[j]$. No substar is used in extending more than one such path. Therefore, the time complexity for constructing all these paths is bounded by

$$O(k_2 n + k_3 n + \cdots + k_n n + n(n-1)) = O((k_2 + k_3 + \cdots k_n + n - 1)n)$$

where k_j is the number of faulty nodes in the substar $S_n[j]$. By our assumption $k_2 + k_3 + \cdots k_n \leq n - 3$. Thus, the time complexity of the algorithm **Parallel-Routing-Star** is bounded by $O(n^2)$. □

Theorem D.1 shows that in the n-star network S_n with at most $n-3$ faulty nodes, for any non-faulty node u, we can construct $\min\{deg_f(u), deg_f(\varepsilon)\}$ node-disjoint fault-free paths from u to ε such that the length of the paths is bounded by $dist(u) + 8$. The following example shows that the bound on the path length in the theorem is actually almost optimal.

Consider the n-star network S_n. Let the source node be $u = (21)$. Here we have omitted the trivial cycles in the cycle structure. Then $dist(u) = 1$. Suppose that all neighbors of u and all neighbors of ε are non-faulty. By Theorem D.1, there are

$n-1$ node-disjoint fault-free paths from u fro ε. Thus, for each i, $3 \le i \le n$, the edge $[u, u_i]$ leads to one P_i of these node-disjoint paths from u fro ε, where $u_i = (i21)$. Note that the edge $[u, u_i]$ does not follow the Shortest Path Rules. Now suppose that the node $(i2)(1)$ is faulty, for $i = 3, 4, \ldots, n-1$ (so there are $n-3$ faulty nodes). Then the third node on the path P_i must be $v_i = (ji21)$ for some $j \neq 1, 2, i$, and the edge $[u_i, v_i]$ does not follow the Shortest Path Rules. Since the only edge from v_i that follows the Shortest Path Rules is the edge $[v_i, u_i]$, the next edge $[v_i, w_i]$ on P_i again does not follow the Shortest Path Rules. Now since all the first three edges on P_i do not follow the Shortest Path Rules, by Fact 2.2, $dist(w_i) = dist(u) + 3 = 4$, and the path P_i needs at least four more edges to reach ε. That is, the length of the path P_i is at least $7 = dist(u) + 6$. Thus, with $n-3$ faulty nodes, among the $n-1$ node-disjoint paths from u to ε, at least $n-3$ of them must have length larger than or equal to $dist(u) + 6$.

The situation given above seems a little special since the distance $dist(u)$ from u to ε is very small. In fact, even for large distance nodes u, we can still construct many examples in which some of the node-disjoint fault-free paths connecting u and ε must have length at least $dist(u) + 6$. For example, let $u = (a_1 a_2 \cdots a_{n-3} 1)$, where $\langle a_1 a_1 \cdots a_{n-3} \rangle$ is any permutation of any $n-3$ symbols in $\{2, 3, \ldots, n\}$. Let $i \notin \{a_1, \ldots, a_{n-3}, 1\}$, and suppose that the nodes $(ia_1)(a_2 \cdots a_{n-3} 1)$, $(ia_1 a_2)(a_3 \cdots a_{n-3} 1)$, $\ldots$, $(ia_1 \cdots a_{n-3})(1)$ are faulty. Then $deg_f(u) = deg_f(\varepsilon) = n-1$ and there are $n-1$ node-disjoint paths from u to ε. Similar to the analysis given above, we can verify that the path from u to ε whose second node is $(ia_1 \cdots a_{n-3} 1)$ must have length larger than or equal to $dist(u) + 6$.

E. Chapter Summary

In this chapter, we have demonstrated that the star networks are strongly fault tolerant. We have presented an algorithm of running time $O(n^2)$ that for two given non-faulty nodes u and v of n-star network with at most $n-3$ faulty nodes, constructs the maximum number (i.e., $\min\{deg_f(u), deg_f(v)\}$) of node-disjoint fault-free paths from u to v such that the length of the paths is bounded by $dist(u,v)+8$. We have shown that the time complexity of our algorithm is optimal, and the length of the paths constructed by our algorithm is almost optimal. Moreover, our algorithm does not require prior knowledge of the failures: in a single round communication, the algorithm can find out the faulty neighbors of the nodes u and v, then the algorithm constructs the node-disjoint paths and avoids faulty nodes whenever they are encountered during the routing. Finally, the study of strong fault tolerance shows another advantage of the star networks over the popular hypercube networks. In particular, the orthogonal partition of the star networks makes the construction of node-disjoint paths very convenient while other popular network topologies, such as the hypercube networks, do not seem to have this nice decomposition structure.

CHAPTER III

STRONG FAULT TOLERANCE OF THE HYPERCUBE NETWORKS

A. Chapter Overview

In the previous chapters, the concept of *the strong fault tolerance* was introduced and studied for the star networks. In this chapter, we continue the study of the strong fault tolerance for the hypercube networks.

The study of strong fault tolerance in the star networks showed that node-disjoint paths can be constructed efficiently based on the *orthogonal partition* of the star networks with faults, which decomposes the n-star network into $n-1$ $(n-1)$-dimensional substar networks and an independent set I of $(n-1)!$ nodes. Roughly speaking, a path from a non-faulty neighbor of the source node u to a non-faulty neighbor of the destination node v is constructed in a separated $(n-1)$-dimensional substar, and the independent set I helps the paths to enter the substar from a proper node.

We observe that the techniques used in studying star networks are not applicable to the case for hypercube networks. Specifically, the hypercube networks do not seem to have similar orthogonal decomposition structure. Parallel routing in the n-dimensional hypercube networks may require constructing n node-disjoint paths, while an n-dimensional hypercubes can be decomposed into at most n $(n-1)$-dimensional subcubes. Therefore, there may be no extra nodes available that can help to distribute the paths into the subcubes.

We develop new techniques that construct node-disjoint paths between pairs of neighbors of the source node u and the destination node v in a hypercube network Q_n. First, a prematching process pairs non-faulty neighbors of u and v in Q_n. For given pairs of neighbors of u and v, we introduce three procedures to construct paths by permutations of edge sequences between them. Node-disjoint paths are constructed by searching proper paths, ensuring that each node in a path is not used by other paths. Our algorithm constructs node-disjoint paths in optimal time, and the length of the paths is also optimal in the hypercube network Q_n: For any two non-faulty nodes u and v in Q_n with at most $n-2$ faulty nodes, the algorithm constructs $\min\{deg_f(u), deg_f(v)\}$ node-disjoint paths of minimum length plus 4 between u and v in time $O(n^2)$.

B. Properties of Hypercube Networks

An n-dimensional hypercube Q_n is an undirected graph consisting of 2^n nodes represented by binary numbers from 0 to $2^n - 1$, and $n2^{n-1}$ edges connecting nodes whose binary representations differ by exactly one bit. An edge is called an *i-edge* if two nodes connected by it differ in the ith bit (the first bit is the leftmost bit). The *Hamming distance* between two nodes u and v, $dist(u,v)$ is the length of the shortest path from u to v. Actually, $dist(u,v)$ is the number of bits in which the binary representations of u and v differ. Since the hypercube Q_n is vertex-symmetric, a set of node-disjoint paths from a node u' to a node v' can be mapped to a set of node-disjoint paths from the node $u = 1^r0^{n-r}$ to the node $v = 0^n$ in a straightforward way, where $r = dist(u,v)$. Therefore, we will concentrate on the construction of node-disjoint paths from the node u of the form 1^r0^{n-r} to the node v of the form 0^n in Q_n.

The node connected from the node u by an i-edge is denoted by u_i, and the node connected from the node u_i by a j-edge is denoted by $u_{i,j}$. A path P from the node $u = 1^r0^{n-r}$ to the node $v = 0^n$ can be uniquely specified by a sequence of labels of the edges on P in the order of traversal. In particular, a path from the node u to the node v that uses an i_1-edge, an i_2-edge, …, an i_r-edge, in that order, will be denoted by $u\langle i_1, i_2, \ldots, i_r\rangle v$. For example, for the nodes $u = 111100$ and $v = 000000$, $u\langle 3, 1, 4, 2\rangle v$ specifies the path $111100 \rightarrow 110100 \rightarrow 010100 \rightarrow 010000 \rightarrow 000000$. We extend this notation for a single label sequence to a set of label sequences, as follows. Let S be a set of label sequences, then the notation $u\langle\ S\ \rangle v$ denotes the set of paths:

$$u\langle\ S\ \rangle v = \{\ u\langle j_1, j_2, \ldots, j_r\rangle v \mid (j_1, j_2, \ldots, j_r) \text{ is a label sequence in } S\ \}$$

For example, suppose $S = \{(3, 1, 4, 2), (1, 4, 2, 3), (4, 2, 3, 1), (2, 3, 1, 4)\}$, then $u\langle\ S\ \rangle v$ consists of four paths from u to v: $u\langle 3, 1, 4, 2\rangle v$, $u\langle 1, 4, 2, 3\rangle v$, $u\langle 4, 2, 3, 1\rangle v$, and $u\langle 2, 3, 1, 4\rangle v$.

We say that an edge $[w_1, w_2]$ *does not lead to a shortest path* (from w_1) to a node w_3 if $dist(w_1, w_3) \leq dist(w_2, w_3)$.

Fact B.1 *If an edge $[w_1, w_2]$ in Q_n does not lead to a shortest path to w_3, then $dist(w_2, w_3) = dist(w_1, w_3) + 1$. In general, if in a path P from a node w_1 to a node w_3, there are exactly k edges that do not lead to a shortest path to w_3, then the length of the path P is equal to $dist(w_1, w_3) + 2k$.*

It is known [52] that for any two nodes u and v in Q_n, there exist n node-disjoint paths such that $dist(u, v)$ of them are of length $dist(u, v)$, and the remaining $n - dist(u, v)$ of them are of length $dist(u, v) + 2$.

C. Case 1: u and v have no Faulty Neighbors

Our parallel routing algorithm is based on an effective pairing of the neighbors of the nodes $u = 1^r 0^{n-r}$ and $v = 0^n$. First, we assume that the nodes u and v have no faulty neighbors. We pair the neighbors of u and v by the following strategy:

Prematch-I

{ Assumption: u and v have no faulty neighbors. }

1. pair u_i with v_{i-1} for $1 \leq i \leq r$ [1] ;
2. pair u_j with v_j for $r+1 \leq j \leq n$;

Under the pairing given by **Prematch-I**, we construct parallel paths between the paired neighbors of u and v using the following procedure:

Procedure-I

1. For $1 \leq i \leq r$, and the paired neighbors u_i and v_{i-1}, we construct $n-2$ node-disjoint paths between u_i and v_{i-1}, which consist of $r-2$ paths

$$u_i \langle\ S_1\ \rangle v_{i-1}, \tag{3.1}$$

where S_1 is the set of all cyclic permutations of the sequence $(i+1, \ldots, r, 1, \ldots, i-2)$, plus $n-r$ paths of the form

$$u_i \langle\ h, i+1, \ldots, r, 1, \ldots, i-2, h\ \rangle v_{i-1}, \tag{3.2}$$

for all h, $r+1 \leq h \leq n$.

2. For $r+1 \leq j \leq n$, and the paired neighbors u_j and v_j, we construct $n-1$ node disjoint paths between u_j and v_j, which consist of r paths

$$u_j \langle\ S_2\ \rangle v_j \tag{3.3}$$

[1]The calculation for indices between 1 and r can be given by a rather lengthy formula based on modular operation. For simplicity, we only need to remember the following three special cases: Let i be an index between 1 and r. (1) for $i = 1$, $i-1$ is interpreted as r and $i-2$ is interpreted as $r-1$; (2) for $i = 2$, $i-2$ is interpreted as r; and (3) for $i = r$, $i+1$ is interpreted as 1.

where S_2 is the set of all cyclic permutations of the sequence $(1, 2, \ldots, r)$, plus $n - r - 1$ paths of the form

$$u_j\langle\ h, 1, 2, \ldots, r, h\ \rangle v_j, \tag{3.4}$$

for all $h \neq j$, and $r + 1 \leq h \leq n$.

The paths constructed by cyclic permutations of a sequence are pairwise disjoint. Thus, the paths constructed in (3.1) for each pair of neighbors u_i and v_{i-1} are pairwise disjoint. The paths constructed for u_i and v_{i-1} in (3.2) are also pairwise disjoint because each contains a unique label h. Finally, since each path in (3.2) has the bit h flipped, where $r + 1 \leq h \leq n$, it must be node-disjoint with any path in (3.1). In conclusion, the $n - 2$ paths in (3.1) and (3.2) constructed for the neighbors u_i and v_{j-1} must be node-disjoint. Similarly, we can verify that the $n - 1$ paths in (3.3) and (3.4) constructed for the neighbors u_j and v_j are also node-disjoint.

For $r = 1$, or $r = n = 2$, or $r = n = 3$, parallel routing is straightforward because it is not difficult to see that a fault-free path between any two non-faulty neighbors of u and v can be always found if all neighbors of u and v are non-faulty. Thus, we assume $n > r$ when $r = 2$ or 3, or $r > 3$. For a path $P = u_i\langle j_1, \ldots j_k, \ldots j_t\rangle v_j$ from u_i to v_j, we define $u_i\langle j_1, \ldots j_k\rangle$ as the node on the path P starting from u_i and following the edge labels in $\langle j_1, \ldots j_k\rangle$.

Lemma C.1 *If a path P_x constructed by* **Procedure-I** *for a pair (u_x, v_y) shares a common node with a path P_s constructed by* **Procedure-I** *for a pair (u_s, v_t), $x \neq s$, then P_x must be of the form $u_x\langle s, \ldots\rangle v_y$.*

PROOF. For two paths P_x and P_s such that P_x is for the pair (u_x, v_y) and P_s is for the pair (u_s, v_t), $x \neq s$, assume that P_x and P_s have a common node $w_0 = u_x\langle \ldots k\rangle =$

$u_s\langle \dots k'\rangle$. By our construction, x does not appear in the sequence $\langle \dots k\rangle$, and s does not appear in the sequence $\langle \dots k'\rangle$. Thus, the xth bit and sth bit of w_0 must be different from that of u. In particular, s and x must appear in the sequences $\langle \dots k\rangle$ and $\langle \dots k'\rangle$, respectively. Thus, we must have $w_0 = u_x\langle \dots s, \dots k\rangle = u_s\langle \dots x, \dots k'\rangle$. We show below that the node w_0 must have the form $u_x\langle s, \dots k\rangle$. In consequence, the path P_x must be of the form $u_x\langle s, \dots\rangle v_y$.

Case 1. $1 \le x \le r$ and $1 \le s \le r$.

Suppose the common node w_0 is of the form $u_x\langle \dots s_0, s, \dots k\rangle = u_s\langle \dots x, \dots k'\rangle$, then s_0 must be either $s-1$ (if $s \ne x+1$), $s-3$ (if $s = x+1$), or h for some $h > r$. If $s_0 = s-1$ then the node $u_s\langle \dots x, \dots k'\rangle$ has the $(s-1)$th bit identical to that of u (note by our construction, $s-1$ does not appear in the sequence $\langle \dots k'\rangle$) while the node $u_x\langle \dots s_0, s, \dots k\rangle$ has the $(s-1)$th bit different from that of u, resulting in a contradiction. If $s_0 = s-3$ and $s = x+1$ then we also get a contradiction because $x = s-1$ and x cannot appear in the sequence $\langle \dots k'\rangle$ for the mode $u_s\langle \dots k'\rangle$. Finally, if $s_0 = h$ then w_0 must be of the form $u_x\langle h, x+1, \dots k\rangle = u_s\langle h, s+1, \dots k'\rangle$, where $\langle x+1, \dots k\rangle$ is a prefix of $\langle x+1, \dots r, 1, \dots x-2, h\rangle$ and $\langle s+1, \dots k'\rangle$ is a prefix of $\langle s+1, \dots r, 1, \dots s-2, h\rangle$. Since $x \ne s$, it is easy to see that this is impossible. Thus, the index $s_0 = h$ is impossible, and w_0 must be of the form $w_0 = u_x\langle s, \dots k\rangle$.

Case 2. $1 \le x \le r$ and $r+1 \le s \le n$, or $r+1 \le x \le n$ and $1 \le s \le r$.

First assume $1 \le x \le r$ and $r+1 \le s \le n$. The sequence in the path P_x must be of the form $u_x\langle h, x+1, \dots, h\rangle v_{x-1}$ for some $h > r$ since $s > r$. Since h is the only index larger than r in this sequence and $s > r$, we must have $h = s$. Thus, the path P_x must be of the form $u_x\langle s, \dots k, \dots\rangle v_{x-1}$. Now assume $r+1 \le x \le n$ and $1 \le s \le r$. Then the common node w_0 must be of the form $u_x\langle \dots s, \dots k\rangle = u_s\langle x, s+1, \dots k'\rangle$. Suppose $w_0 = u_x\langle \dots s_0, s \dots k\rangle$, then s_0 must be either $s-1$ or h for some $h > r$. If $s_0 = s-1$ then it makes a contradiction because $s-1$ does not appear in the sequence

$\langle x, s+1, \dots k' \rangle$. If $s_0 = h$, then since $x, h > r$, $u_x\langle \dots s_0, s, \dots k \rangle$ has at least two bits higher than r(the xth and the hth) different from u, while $u_s\langle x, s+1, \dots k' \rangle$ has only 1 bit higher than r(the xth) different from u. This would be a contradiction. Thus, w_0 must be of the form $w_0 = u_x\langle s, \dots k \rangle$.

Case 3. $r + 1 \leq x \leq n$ and $r + 1 \leq s \leq n$.

The sequences in P_x and P_s cannot be a permutation of $(1, \dots r)$ because x, $s > r$ and $x \neq s$. Thus, w_0 must be of the form $u_x\langle s, \dots k \rangle = u_s\langle x, \dots k' \rangle$.

Combining all these, we complete the proof. □

Corollary C.2 *Let (u_x, v_y) and (u_s, v_t) be two pairs given by* **Prematch-I**. *Then there is at most one path in the path set constructed by* **Procedure-I** *for the pair (u_x, v_y) that shares common nodes with a path in the path set constructed by* **Procedure-I** *for the pair (u_s, v_t).*

Proof. We have shown that in Lemma C.1, if a path P_x constructed by **Procedure-I** for a pair (u_x, v_y) shares a common node with a path P_s constructed by **Procedure-I** for a pair (u_s, v_t), $x \neq s$, then P_x must be of the form $u_x\langle s, \dots \rangle v_y$. It implies that when P_x and P_s have common nodes, then the forms of P_x and P_s are uniquely decided, i.e., at most one path P_x for (u_x, v_y) has common nodes with a path P_s constructed for (u_s, v_t). □

Corollary C.3 *For a pair (u_i, v_{i-1}), $1 \leq i \leq r$ given by* **Prematch-I**, *a path of the form $u_i\langle i+1, \dots r, 1, \dots i-2 \rangle v_{i-1}$ has no common nodes with any other paths constructed by* **Procedure-I**.

Proof. Suppose the path P_i of the form $u_i\langle i+1, \dots r, 1, \dots i-2 \rangle v_{i-1}$ shares a node with other paths, then by Lemma C.1, the node must be of the form $u_{i+1}\langle i, \dots \rangle$. However, by our construction, no path for the pair (u_{i+1}, v_i) is of the form $u_{i+1}\langle i, \ldots\rangle v_i$. $\square$

We have shown that for each paired nodes by **Prematch-I**, the algorithm **Procedure-I** constructs at least $n-2$ node-disjoint paths between them. Since there may be up to $n-2$ faulty nodes, in the worst case, there can be a pair (u_i, v_{i-1}) of nodes by **Prematch-I** where $1 \leq i \leq r$, for which all $n-2$ paths constructed by **Procedure-I** are blocked. Note that between a pair (u_j, v_j) with $r+1 \leq j \leq n$, **Procedure-I** constructs $n-1$ node-disjoint paths. In this case, we pair the neighbors of u and v by the following rule:

Prematch-II

1. u_i is paired with v_{i-2};
2. u_{i-1} is paired with v_i;
3. u_{i+1} is paired with v_{i-1};
4. For other neighbors of u and v, use **Prematch-I**

For each pair constructed by **Prematch-II**, we construct a path as follows.

Procedure-II

{ Assumption: there is a pair (u_i, v_{i-1}), $1 \leq i \leq r$ given by **Prematch-I** such that all $n-2$ paths constructed by **Procedure-I** for (u_i, v_{i-1}) are blocked by faulty nodes. }

1. For a pair (u_i, v_{i-2}), the path $P_i = u_i\langle i-1, i+1, \ldots, r, 1, \ldots, i-3\rangle v_{i-2}$;
2. For a pair (u_{i-1}, v_i), the path $P_{i-1} = u_{i-1}\langle i+1, i+2, \ldots r, 1, \ldots, i-2\rangle v_i$;

3. For a pair (u_{i+1}, v_{i-1}), the path $P_{i+1} = u_{i+1}\langle i+2, \ldots, r, 1, \ldots, i-2, i\rangle v_{i-1}$;

4. For other pairs, construct paths as follows: For pair (u_g, v_{g-1}), $g \neq i-1, i, i+1$, $1 \leq g \leq r$, the path $P_g = u_g\langle g+1, \ldots, r, 1, \ldots, g-2\rangle v_{g-1}$; For pair (u_j, v_j), $r+1 \leq j \leq n$, the path $P_j = u_j\langle 2, 3, \ldots, r, 1\rangle v_j$ if $i = 1$, and $P_j = u_j\langle 1, 2, \ldots, r\rangle v_j$ if $i \neq 1$.

Lemma C.4 *Suppose that the hypercube Q_n contains at most $n-2$ faulty nodes, and that all $n-2$ paths constructed by* **Procedure-I** *for the pair (u_i, v_{i-1}), where $1 \leq i \leq r$, are blocked by faulty nodes, then the algorithm* **Procedure-II** *constructs n fault-free parallel paths of length bounded by $dist(u, v) + 2$ from u to v.*

PROOF. It easy to see that Paths constructed by **Procedure-II** have length bounded by $dist(u, v) + 2$. In fact, except paths of form $u_j\langle \ldots\rangle v_j$, $r+1 \leq j \leq n$, whose length is $dist(u, v) + 2$, other paths have length $dist(u, v)$.

First, we show that all n paths constructed by **Procedure-II** are fault-free. After that, we will show that these n paths are node-disjoint. Denote the set of the $n-2$ paths constructed by **Procedure-I** for the pair (u_i, v_{i-1}) by F_i. Since Q_n has at most $n-2$ faulty nodes, every faulty node is on a path in F_i.

The path $P_i = u_i\langle i-1, i+1, \ldots, r, 1, \ldots, i-3\rangle v_{i-2}$ and the paths in F_i only share the node u_i because every node in a path in F_i has its $(i-1)$th bit identical to that of u while nodes except u_i in P_i have the $(i-1)$th bit different that of u. Since every faulty node is on a path in F_i and u_i is non-faulty, the path P_i is fault-free. The path $P_{i-1} = u_{i-1}\langle i+1, \ldots, r, 1, \ldots, i-2\rangle v_i$ and the paths in F_i have no common nodes because ith bits in nodes in P_{i-1} and in nodes in paths in F_i are different. Thus, the path P_{i-1} is fault-free. The path $P_{i+1} = u_{i+1}\langle i+2, \ldots, r, 1, \ldots, i-2, i\rangle v_{i-1}$

and the paths in F_i only share the node v_{i-1} because nodes in P_{i+1} except v_{i-1} ($= u_{i+1}\langle i+2,\ldots,i-2,i\rangle$) have the ith bit identical to that of u while nodes in the paths in F_i have the ith bit different from that of u. Since v_{i-1} is non-faulty, the path P_{i+1} is also fault-free. A path of form $P_g = u_g\langle g+1,\ldots,r,1,\ldots,g-2\rangle v_{g-1}$, where $1 \le g \le r$ and $g \ne i-1, i, i+1$, has no common nodes with any paths in F_i by Lemma C.1 (since $g+1 \ne i$). Thus, the path P_g is fault-free. Finally, consider a path P_j constructed for the pair (u_j, v_j), $r+1 \le j \le n$. If $i = 1$, all faulty nodes are in paths between u_1 and v_r and P_j is of the form $u_j\langle 2,3,\ldots,r,1\rangle v_j$. Therefore, all nodes in paths in F_i have their first bit different from that of u while all nodes in the path P_j (except v_j) have their first bit identical to that of u. Since $j > r$, the path P_j has no common nodes with the paths in F_i. Thus, P_j is fault-free. In case $i \ne 1$, P_j is of the form $u_j\langle 1,2,\ldots,r\rangle v_j$, and P_j has no common nodes with any paths in F_i by Lemma C.1 (since $i \ne 1$). This, again, shows that P_j is fault-free.

Therefore, all paths constructed by **Procedure-II** are fault-free.

Now we show that paths constructed by **Procedure-II** are node-disjoint.

It is easy to see that P_i and P_{i-1} have no common nodes because of the index i. Similarly, P_{i-1} and P_{i+1} have no common nodes because of the index $i-1$, and P_i and P_{i+1} have no common nodes because of the index i. Thus, paths P_{i-1}, P_i, and P_{i+1} are disjoint. Moreover, two paths of the forms $u_g\langle g+1,\ldots r,1,\ldots g-2\rangle v_{g-1}$ and $v_{g'}\langle g'+1,\ldots r,1,\ldots g'-2\rangle v_{g'-1}$, where $1 \le g \ne g' \le r$, and $g, g' \ne i-1, i, i+1$, are node-disjoint by Lemma C.1. Moreover, it is easy to see that two paths of the forms $u_j\langle\propto\rangle v_j$ and $u_{j'}\langle\propto\rangle v_{j'}$, where $r+1 \le j \ne j' \le n$ and $\propto$ is either $\langle 1,2,\ldots r\rangle$ or $\langle 2,3,\ldots r,1\rangle$ are node-disjoint, and that a path of the form $u_j\langle\propto\rangle v_j$, where $r+1 \le j \le n$ and $\langle\propto\rangle$ is either $\langle 1,2,\ldots r\rangle$ or $\langle 2,3,\ldots r,1\rangle$, is node-disjoint with the paths P_{i-1}, P_i, and P_{i+1}, and with a path of the form $u_g\langle g+1,\ldots r,1,\ldots g-2\rangle v_{g-1}$, $1 \le g \le r$, $g \ne i-1, i, i+1$. What remains is to show that a path of the form

$u_g\langle g+1,\ldots r,1,\ldots g-2\rangle v_{g-1}$, where $1 \le g \le r$, $g \ne i-1, i, i+1$ is node-disjoint with the paths P_{i-1}, P_i, and P_{i+1}.

Suppose $g \ne i-2$. Since $g \ne i-1, i, i+1$, the path P_i must be of the form $P_i = u_i\langle \ldots g_0, g, \ldots\rangle v_{i-2}$ and $g_0 = g-1$. Thus, P_i and a path of the form $u_g\langle g+1,\ldots r,1,\ldots g-2\rangle v_{g-1}$ have no common nodes because of the index $g-1$. Also, the path P_{i-1} must be of the form $P_{i-1} = u_{i-1}\langle \ldots g_0, g, \ldots\rangle v_i$ and $g_0 = g-1$. Thus, P_{i-1} and a path of the form $u_g\langle g+1,\ldots r,1,\ldots g-2\rangle v_{g-1}$ have no common nodes because of the index $g-1$. Finally, the path P_{i+1} must either be of the form $P_{i+1} = u_{i+1}\langle \ldots g_0, g, \ldots\rangle v_{i-1}$ and $g_0 = g-1$ when $g \ne i+2$, or $P_{i+1} = u_{i+1}\langle g, \ldots\rangle v_{i-1}$ $(= u_{g-1}\langle g, \ldots\rangle v_{i-1})$ when $g = i+2$. For both cases, they have no common nodes with a path of form $u_g\langle g+1,\ldots r,1,\ldots g-2\rangle v_{g-1}$ because of the index $g-1$. Now, consider the case $g = i-2$. Since $g = i-2$, the path P_i from u_{g+2} to v_g and a path of the form $u_g\langle g+1,\ldots r,1,\ldots g-2\rangle v_{g-1}$ have no common nodes because of the index g. Also, all nodes in the path $P_{i-1} = u_{g+1}\langle \ldots, g\rangle v_{g+2}$ have their gth bit identical to that of u(except v_{g+2}) while all nodes in a path of the form $u_g\langle g+1,\ldots r,1,\ldots g-2\rangle v_{g-1}$ have their gth bit different from that of u. Nodes v_{g-1} and v_{g+2} are identical only when $r = 3$. However, if $r = 3$, then **Procedure-II** is not executed. Finally, all nodes in the path P_{i+1} from u_{g+3} to v_{g+1} have their $(g+1)$th bit identical to that of u while all nodes in a path of the form $u_g\langle g+1,\ldots r,1,\ldots g-2\rangle v_{g-1}$ except u_g have their $(g+1)$th bit different from that of u. Similarly, $u_g(v_{g-1})$ and $u_{g+3}(v_{g+1})$ are identical only when $r = 3(r = 2)$, respectively. Thus, a path of form $u_g\langle g+1,\ldots r,1,\ldots g-2\rangle v_{g-1}$, where $1 \le g \le r$, $g \ne i-1, i, i+1$ is node-disjoint with the paths P_{i-1}, P_i, and P_{i+1}.

Therefore, all paths constructed by **Procedure-II** are pairwise node-disjoint. □

We summarize all above discussions in the algorithm called **Parallel-Routing-Cube-I**. The algorithm **Parallel-Routing-Cube-I** is given in Fig. 3.

Algorithm. Parallel-Routing-Cube-I

{Assumption: u and v have no faulty neighbors. }

Input: non-faulty nodes $u = 1^r 0^{n-r}$ and $v = 0^n$ in Q_n with at most $n-2$ faulty nodes.

Output: $\min\{deg_f(u), deg_f(v)\}$ parallel fault-free paths of length $\leq dist(u,v)+4$ from u to v.

for each pair (u_i, v_j) given by **Prematch-I do**

1. **if** all paths for (u_i, v_j) by **Procedure-I** include faulty nodes
   then use **Prematch-II** and **Procedure-II** to construct n parallel
   paths from u to v;
   STOP.
2. **if** there is a fault-free unused path from u_i to v_j by **Procedure-I**
   then mark the path as used by (u_i, v_j);
3. **if** all fault-free paths constructed for (u_i, v_j) include used nodes
   if $n = r = 4$
   then construct 4 paths between u and v by cyclic permutations
   of a sequence $(1, 3, 2, 4)$ when $i = 3$ or a sequence $(2, 4, 3, 1)$
   when $i = 4$;
   else
   pick the first fault-free path P for (u_i, v_j), and for the pair
   $(u_{i'}, v_{j'})$ that uses a node on P, find a new path;

Fig. 3. Parallel routing on the hypercube network with faulty nodes when u and v have no faulty neighbors

Lemma C.4 guarantees that step 1 of the algorithm **Parallel-Routing-Cube-I** constructs n fault-free parallel paths of length $\leq dist(u,v)+2$ from u to v. Step 3 of the algorithm requires further explanation. In particular, we need to show that for the pair $(u_{i'}, v_{j'})$, we can always construct a new fault-free path from $u_{i'}$ to $v_{j'}$ in which no nodes are used by other paths. This is ensured by the following lemma.

Lemma C.5 *Let (u_i, v_j) and $(u_{i'}, v_{j'})$, $i' < i$ be two pairs given by* **Prematch-I** *such that two paths constructed for (u_i, v_j) and $(u_{i'}, v_{j'})$ share a node. Then the algorithm* **Parallel-Routing-Cube-I** *can always find fault-free paths for (u_i, v_j) and $(u_{i'}, v_{j'})$, in which no nodes are used by other paths.*

Proof. We prove this lemma based on the assumption that step 3 of **Parallel-**

Routing-Cube-I can be invoked once during the execution. This assumption will be proved in the following lemma. We assume that we will search a fault-free and unused path for each pair given by **Prematch-I** in order of cyclic permutations as given in **Procedure-I** in the following discussion. For example, $u_1, \langle 2, \ldots \rangle v_r$, $u_1\langle 3, \ldots \rangle v_r$, $\cdots$, $u_2\langle 3, \ldots \rangle v_1$.

Case 1. $1 \leq i \leq r$.

If $r \leq 3$, then we can always find a fault-free and unused path for the pair (u_i, v_{i-1}), $1 \leq i \leq r$. Thus, we assume $r \geq 4$.

Case 1.1. $i = 1$ or 2.

If $i = 1$ or 2, then either we can find fault-free and unused paths for the pair (u_1, v_r) and (u_2, v_1) because the path sets for pairs (u_1, v_r) and (u_2, v_1) do not share common nodes, or use **Prematch-II** and **Procedure-II** when all paths for (u_1, v_r) or (u_2, v_1) include faulty nodes.

Case 1.2. $3 \leq i \leq r - 1$ when $1 \leq i' \leq i - 2$ or $i = r$ when $2 \leq i' \leq i - 2$.

Suppose $u_i\langle i', \ldots \rangle v_{i-1}$ is the first used path we found in step 3 of **Parallel-Routing-Cube-I**. Then paths $u_{i'}\langle b, \ldots \rangle v_{i'-1}$, $i' + 1 \leq b \leq i - 1$, must be faulty. Otherwise, $u_i\langle i', \ldots \rangle v_{i-1}$ would not contain nodes used on the path $u_{i'}\langle i, \ldots \rangle v_{i'-1}$ because $u_{i'}\langle b, \ldots \rangle v_{i'-1}$, $i' + 1 \leq b \leq i - 1$ would be fault-free and unused. Thus, there are at least $i - i' - 1$ faulty paths for pair $(u_{i'}, v_{i'-1})$. Since $u_i\langle i', \ldots \rangle v_{i-1}$ is the first used path, paths $u_i\langle g, \ldots \rangle v_{i-1}$, $1 \leq g \leq i' - 1$, are faulty. Also, all other unused paths of the form $u_i\langle g, \ldots \rangle v_{i-1}$, $g > i$, should be faulty if we execute step 3. There are $n - i$ such paths. Since only the path $u_{i'}\langle i, \ldots \rangle v_{i'-1}$ for pair $(u_{i'}, v_{i'-1})$ shares common nodes with a path constructed for pair (u_i, v_{i-1}), we already detect at least $(i - i' - 1) + (i' - 1) + (n - i) = n - 2$ faulty nodes in the hypercube. Thus, if a path of the form $u_{i'}\langle b, \ldots \rangle v_{i'-1}$, $b > i$, exists, it is fault-free and unused. We show that either such a path exists or step 3 of **Parallel-Routing-Cube-I** is not executed unless $n = r = 4$.

If $n > r$, then we can find a fault-free and unused path for the pair $(u_{i'}, v_{i'-1})$ such as $u_{i'}\langle r+1, \ldots\rangle v_{i'-1}$ in step 3. If $n = r > 4$ and $i' \neq i-2$, then step 3 of **Parallel-Routing-Cube-I** is not executed because there exists a fault-free and unused path such as $u_i\langle i'+1, \ldots\rangle v_{i-1}$. Recall that $i' \neq i-1$ because sets of paths constructed for pair (u_i, v_{i-1}) and (u_{i-1}, v_{i-2}) do not share common nodes. In case $n = r > 4$ and $i' = i-2$, we can find a fault-free and unused path for the pair $(u_{i'}, v_{i'-1})$ such as $u_{i'}\langle i+1, \ldots\rangle v_{i'-1}$. Specifically, $i' = i-2$ and $i = n$, then we find $u_{i'=n-2}\langle 1, \ldots\rangle v_{(i'-1)=n-3}$, which is fault-free and unused. In case $n = r = 4$, proof is straightforward.

Case 2. $r + 1 \leq i \leq n$.

We show that step 3 of **Parallel-Routing-Cube-I** cannot be executed, and there exists at least one fault-free and unused path for (u_i, v_i). By way of contradiction, we assume that all fault-free paths constructed for (u_i, v_i) include used nodes and $u_i\langle i', \ldots\rangle v_i$ is fault-free and the first used path we found in step 3.

Case 2.1. $1 \leq i' \leq r$.

If $r = 2$ or 3, then paths constructed for $(u_{i'}, v_{i'-1})$ do not include the index i, and there are no common nodes in paths constructed for the pairs $(u_{i'}, v_{i'-1})$ and (u_i, v_i). Thus, we assume $r \geq 4$.

Suppose $i' = 1$. Then paths $u_1\langle b, \ldots\rangle v_r$, where $2 \leq b \leq r-1$ and $r+1 \leq b \leq i-1$, are faulty. Otherwise, they are fault-free and unused, and $u_i\langle 1, \ldots\rangle v_i$ would not contain nodes used on the path $u_1\langle i, \ldots\rangle v_r$. Thus, there are at least $(r-2)+(i-r-1) = i-3$ faulty paths for the pair (u_1, v_r). Since we execute step 3, all other unused paths $u_i\langle g, \ldots\rangle v_i$, $g > i$ must be faulty. Thus, we already detect at least $(i-3)+(n-i) = n-3$ faulty nodes in the hypercube. In addition, the path $u_i\langle 2, \ldots\rangle v_i$ must be faulty. Otherwise, $u_2\langle i, \ldots\rangle v_1$ is used, and paths $u_2\langle b, \ldots\rangle v_1$, $3 \leq b \leq r$, also should be faulty. Since $r \geq 4$, there are at least two such paths. It yields at least $n-1$ faulty nodes and contradicts the assumption that there are at least $n-2$ faulty nodes in the hypercube. Now, since $r \geq 4$ and we already detect at least $n-2$ faulty nodes, the path $u_i\langle 3, \ldots, \rangle v_i$ should be faulty-free. Also, it cannot include nodes used by the path $u_3\langle i, \ldots\rangle v_2$ because it means that the path $u_3\langle 4, \ldots\rangle v_2$ is also faulty. It shows a contradiction of the assumption that all fault-free paths constructed for (u_i, v_i) include used nodes.

Suppose $i' = 2$. Then paths $u_2\langle b, \ldots\rangle v_1$, where $3 \leq b \leq i-1$, and paths $u_i\langle g, \ldots\rangle v_i$, where $g = 1$ and $g > i$, should be faulty. Thus, we already detect at least $(i-3)+1+(n-i) = n-2$ faulty nodes. Since $r \geq 4$, paths $u_i\langle g, \ldots\rangle v_i$, $3 \leq g \leq r$, are fault-free and unused. It again manifests contradiction.

Suppose $3 \leq i' \leq r$. Then paths $u_{i'}\langle b, \ldots\rangle v_{i'-1}$, $1 \leq b \leq i'-2$, are faulty or used. Suppose a path of the form $u_{i'}\langle b, \ldots\rangle v_{i'-1}$, $1 \leq b \leq i'-2$, is used, then paths of the form $u_b\langle b+1, \ldots, r, 1, \ldots, b-2\rangle v_{b-1}$ must be faulty. Otherwise, it is faulty-free and unused, and $u_{i'}\langle b, \ldots\rangle v_{i'-1}$ would not be used. Recall that a path of the form $u_b\langle b+1, \ldots, r, 1 \ldots, b-2\rangle v_{b-1}$ has no common nodes with other paths constructed by **Procedure-I**. Also, paths $u_{i'}\langle b, \ldots\rangle v_{i'-1}$, $i'+1 \leq b \leq i-1$, and paths $u_i\langle g, \ldots\rangle v_i$, where $1 \leq g \leq i'-1$ and $g > i$, are faulty. Since $g \neq i'$ and $i' \geq 3$, it yields at least $(i'-2)+(i-i'-1)+(i'-1)+(n-i) = n-1$ faulty nodes. Thus, the path $u_i\langle i', \ldots\rangle v_i$ itself is fault-free and unused. Therefore, we can find a fault-free and unused path for the pair (u_i, v_i), $r+1 \leq i \leq n$, and step 3 is not executed.

Case 2.2. $r+1 \leq i' \leq n$.

Similarly, since $u_i\langle i', \ldots\rangle v_i$ is the first used path for (u_i, v_i), paths $u_{i'}\langle b, \ldots\rangle v_{i'}$, where $1 \leq b(\neq i') \leq i-1$, are faulty or used. Suppose a path of the form $u_{i'}\langle b, \ldots\rangle v_{i'}$, $1 \leq b \leq r$, is used, then the path $u_b\langle b+1, \ldots r, 1, \ldots, b-2 > v_{b-1}$ should be faulty. Suppose a path of the form $u_{i'}\langle b, \ldots\rangle v_{i'}$, $r+1 \leq b(\neq i') \leq i-1$, is used, then the path $u_b\langle 1,\ldots,r\rangle v_b$ should be faulty. Since a path of the form $u_{i'}\langle b,\ldots\rangle v_{i'}$, $1 \le b(\ne i') \le i-1$, a path of the form $u_b\langle b+1,\ldots,r,1\ldots,b-2\rangle v_{b-1}$, $1 \le b \le r$, and a path of the form $u_b\langle 1,\ldots r\rangle v_b$, $r+1 \le b(\ne i) \le i-1$, have no common nodes, it yields $i-2$ faulty nodes. Also, paths $u_i\langle g,\ldots\rangle v_i$, where $1 \le g \le i'-1$, and $g > i$, are faulty. Since $g \ne i'$, we already detect at least $(i-2)+(i'-1)+(n-i) = n+i'-3$ faulty nodes in Q_n. Since $r \ge 1$, we have $i' \ge 2$, and it yields at least $n-1$ faulty nodes. Thus, $u_i\langle i',\ldots\rangle v_i$ itself should be fault-free and unused. Therefore, step 3 of **Parallel-Routing-Cube-I** is not executed. □

In the above lemma, we assume that step 3 of **Parallel-Routing-Cube-I** can be invoked at most once during the execution. We prove this below.

Lemma C.6 *Step 3 of* **Parallel-Routing-Cube-I** *is invoked at most once during the whole execution.*

Proof. Once step 3 is invoked, from Lemma C.5, there are at least $n-2$ faulty nodes on paths $u_{i'}\langle i,\ldots\rangle v_{i'-1}$, where $i'+1 \le b \le i-1$, and $u_i\langle b,\ldots\rangle v_{i-1}$, where $1 \le b \le i'-1$ and $b > i$. It suffices to show that **Parallel-Routing-Cube-I** constructs a fault-free and unused path for each other pair (u_x, v_y), $x \ne i, i'$ without invoking step 3 again. Recall that i and i' are between 1 and r, and step 3 is not executed when $i > r$.

From corollary C.3, a path of the form $u_g\langle g+1,\ldots,r,1,\ldots,g-2\rangle v_{g-1}$, $1 \le g \le r$ has no common nodes with any other paths constructed by **Procedure-I**. Thus, paths $u_g\langle g+1,\ldots,r,1,\ldots,g-2\rangle v_{g-1}$, $1 \le g \le r$ and $g \ne i, i'$ are fault-free and unused. Therefore, for each pair (u_g, v_{g-1}), $g \ne i, i'$, **Parallel-Routing-Cube-I** constructs a path of the form $u_g\langle g+1,\ldots,r,1,\ldots,g-2\rangle v_{g-1}$. Let $(g, g+1,\ldots,r,1,\ldots,g-1)$, $1 \le g(\ne i, i') \le r$ be a sequence in the path constructed for a pair (u_g, v_{g-1}). Since

$r \geq 4$, there exists at least one such path. For each pair (u_j, v_j), $r+1 \leq j \leq n$, we can construct a path of the form $u_j\langle g, g+1, \ldots, r, 1, \ldots, g-1\rangle v_j$. Since each path includes a unique index j, it is node-disjoint with paths of the form $u_g\langle g+1, \ldots, r, 1, \ldots, g-2\rangle v_{g-1}$. Also, by Lemma C.1, it is node-disjoint with paths constructed for pairs (u_i, v_{i-1}) and $(u_{i'}, v_{i'-1})$ because $g \neq i, i'$. It shows that we can find a fault-free and unused path for all other pairs (u_x, v_y), $x \neq i, i'$. □

D. Case 2: u or v has faulty neighbors

So far, we have assumed that all neighbors of the source node u and the destination node v are non-faulty. Now we relax such a restriction to deal with faulty neighbors of two nodes u and v. If only one of u or v has faulty neighbors, we can use **Parallel-Routing-Cube-I** to find $\min\{deg_f(u), deg_f(v)\}$ parallel paths by a slight modification: We first assume that u and v have no faulty neighbors. Then it can be regarded as case 1 with at most $n-3$ faulty nodes. Thus, for two given nodes u and v that have no faulty neighbors in Q_n with at most $n-3$ faulty nodes, we apply **Parallel-Routing-Cube-I**, and then discard the paths including faulty neighbors of u or v. This will give us $\min\{deg_f(u), deg_f(v)\}$ node-disjoint paths between u and v. Therefore, we assume that both u and v have faulty neighbors. We provide **Prematch-III** to pair the edges incident on the neighbors of the nodes u and v, instead of the neighbors themselves.

Prematch-III

{ Assumption: Both u and v have faulty neighbors.}

for each edge $[u_i, u_{i,i'}]$ where $1 \leq i, i' \leq n$ and $i \neq i'$ **do**

1. if $1 \leq i, i' \leq r$ [2] and $i' = i+1$, then pair $[u_i, u_{i,i'}]$ with the edge $[v_{i-1,i-2}, v_{i-1}]$;
2. if $1 \leq i, i' \leq r$ and $i' = i-1$, then pair $[u_i, u_{i,i'}]$ with the edge $[v_{i'-1,i'-2}, v_{i'-1}]$;
3. otherwise, pair $[u_i, u_{i,i'}]$ with the edge $[v_{j,j'}, v_j]$, where the indices j and j' are such that **Prematch-I** pairs the node $u_{i'}$ with v_j, and the node u_i with $v_{j'}$.
4. if a pair of edges has a faulty node, mark it as faulty.

For a non-faulty node u_i, $1 \leq i \leq n$, and each neighbor $u_{i,i'}$ of u_i, $i \neq i'$, **Prematch-III** can pair the edge $[u_i, u_{i,i'}]$ with the unique edge $[v_{j,j'}, v_j]$. Thus, for a non-faulty node u_i, there are at most $n-1$ pairs of edges with u_i. Also, a node u has at least one faulty neighbor that is not included in edges paired with u_i. Thus, for a non-faulty node u_i, there exist at least two non-faulty pairs of edges with u_i. On the other hand, for a non-faulty node v_j, an edge $[v_{j,j-1}, v_j]$, $1 \leq j \leq r$ can be paired with $[u_{j+1}, u_{j+1,j+2}]$ and $[u_{j+2}, u_{j+2,j+1}]$, where nodes $u_{j+1,j+2}$ and $u_{j+2,j+1}$ are identical. That is, **Prematch-III** can make $n-1$ pairs of edges with v_j such that at most $n-2$ pairs are disjoint. However, since we assume that v has at least one faulty neighbor, there exists a faulty node v_g, $g \neq j$, which is not included in pairs of edges with v_j. Thus, there is at least one non-faulty pair of edges with v_j. Regardless of faulty nodes, Table I shows an example of all possible pairs of edges that can be constructed by **Prematch-III** when $r = 4$, and $n = 6$.

Lemma D.1 *If a pair of edges $p_1 = ([u_x, u_{x,x'}], [v_{y,y'}, v_y])$ given by* **Prematch-III** *shares a common node with a pair of edges $p_2 = ([u_s, u_{s,s'}], [v_{t,t'}, v_t])$ where $x \neq s$ and*

[2]The operations on indices are by mod r.

Table I. Edges paired by **Prematch-III** when $r = 4$ and $n = 6$

	$[u_2, u_{2,3}][v_{1,4}, v_1]$	$[u_3, u_{3,2}][v_{1,4}, v_1]$	$[u_4, u_{4,2}][v_{1,3}, v_1]$	$[u_5, u_{5,2}][v_{1,5}, v_1]$	$[u_6, u_{6,2}][v_{1,6}, v_1]$
$[u_1, u_{1,3}][v_{2,4}, v_2]$		$[u_3, u_{3,4}][v_{2,1}, v_2]$	$[u_4, u_{4,3}][v_{2,1}, v_2]$	$[u_5, u_{5,3}][v_{2,5}, v_2]$	$[u_6, u_{6,3}][v_{2,6}, v_2]$
$[u_1, u_{1,4}][v_{3,2}, v_3]$	$[u_2, u_{2,4}][v_{3,1}, v_3]$		$[u_4, u_{4,1}][v_{3,2}, v_3]$	$[u_5, u_{5,4}][v_{3,5}, v_3]$	$[u_6, u_{6,4}][v_{3,6}, v_3]$
$[u_1, u_{1,2}][v_{4,3}, v_4]$	$[u_2, u_{2,1}][v_{4,3}, v_4]$	$[u_3, u_{3,1}][v_{4,2}, v_4]$		$[u_5, u_{5,1}][v_{4,5}, v_4]$	$[u_6, u_{6,1}][v_{4,6}, v_4]$
$[u_1, u_{1,5}][v_{5,4}, v_5]$	$[u_2, u_{2,5}][v_{5,1}, v_5]$	$[u_3, u_{3,5}][v_{5,2}, v_5]$	$[u_4, u_{4,5}][v_{5,3}, v_5]$		$[u_6, u_{6,5}][v_{5,6}, v_5]$
$[u_1, u_{1,6}][v_{6,4}, v_6]$	$[u_2, u_{2,6}][v_{6,1}, v_6]$	$[u_3, u_{3,6}][v_{6,2}, v_6]$	$[u_4, u_{4,6}][v_{6,3}, v_6]$	$[u_5, u_{5,6}][v_{6,5}, v_6]$	

$y \neq t$, given by **Prematch-III**, *then $x = s'$ and $x' = s$.*

Proof. For two pairs of edges $p_1 = ([u_x, u_{x,x'}], [v_{y,y'}, v_y])$ and $p_2 = ([u_s, u_{s,s'}], v_{t,t'}, v_t])$, $x \neq s$ and $y \neq t$, assume that p_1 and p_2 have a common node w_0. Then $w_0 = u_{x,x'} = u_{s,s'}$, or $w_0 = v_{y,y'} = v_{t,t'}$.

Case 1. $w_0 = u_{x,x'} = u_{s,s'}$.

Suppose the common node w_0 is $u_{x,x'} = u_{s,s'}$, then $u_{x,x'}$ is also identical to $u_{s',s}$ because $u_{s,s'} = u_{s',s}$. Since we assume that $x \neq s$, we have $x = s'$ and $x' = s$.

Case 2. $w_0 = v_{y,y'} = v_{t,t'}$.

Suppose the common w_0 is $v_{y,y'} = v_{t,t'}$, then $y = t'$ and $y' = t$ because we assume that $y \neq t$. If $1 \leq x, x' \leq r$ and $x' = x + 1$, then the edge $[u_x, u_{x,x'}]$ is paired with the edge $[v_{y,y'}, v_y] = [v_{x-1,x-2}, v_{x-1}]$ by **Prematch-III**. However, by our construction, no edge paired by **Prematch-III** is of the form $[v_{x-2,x-1}, v_{x-2}]$. If $1 \leq x, x' \leq r$ and $x' = x - 1$, then the edge $[u_x, u_{x,x'}]$ is paired with the edge $[v_{y,y'}, v_y] = [v_{x'-1,x'-2}, v_{x'-1}]$ by **Prematch-III**. Again, by our construction, no edge paired by **Prematch-III** is of the form $[v_{x'-2,x'-1}, v_{x'-2}]$. For other cases, the edge $[u_x, u_{x,x'}]$ is paired with the edge $[v_{y,y'}, v_y]$ where the indices y and y' are such that **Prematch-I** pairs $u_{x'}$ and v_y, and pairs u_x and $v_{y'}$. Consider an edge paired with the edge $[v_{y',y}, u_{y'}]$. The edge $[v_{y',y}, v_{y'}]$ is paired with the edge $[u_s, u_{s,s'}]$ where the indices y' and y are such that

Prematch-I pairs $u_{s'}$ and $v_{y'}$, and u_s and v_y. Thus, we have $x = s'$ and $x' = s$. $\square$

A pair of edges $([u_i, u_{i,i'}], [v_{j,j'}, v_j])$ given by **Prematch-III** can be identified by the first two indices i and i' in an edge with a neighbor u_i of u. Thus, if we represent all pairs of edges with u_i in a column i of a matrix, then we need only the index i' to identify each pair of edges with u_i. Also, each pair of edges with a node u_i includes a unique node v_j. Thus, for each pair of edges $([u_i, u_{i,i'}], [v_{j,j'}, v_j])$, we can represent it by using the index i' in row j and column i of a matrix. From this observation, we represent edges paired by **Prematch-III** as a matrix $M = [e_k]$ such that for an edge $[u_i, u_{i,i'}]$, an entry e_k in row j and column i is,

$$e_k = \begin{cases} i' & \text{if the edge } [u_i, u_{i,i'}] \text{ is paired with the edge } [v_{j,j'}, v_j] \text{ by } \textbf{Prematch-III}, \\ 0 & \text{otherwise} \end{cases}$$

The following matrix M represents pairs of edges shown in Table I.

$$M = \begin{bmatrix} 0 & 3 & 2 & 2 & 2 & 2 \\ 3 & 0 & 4 & 3 & 3 & 3 \\ 4 & 4 & 0 & 1 & 4 & 4 \\ 2 & 1 & 1 & 0 & 1 & 1 \\ 5 & 5 & 5 & 5 & 0 & 5 \\ 6 & 6 & 6 & 6 & 6 & 0 \end{bmatrix} \tag{3.5}$$

For example, an edge pair $([u_3, u_{3,4}], [v_{2,1}, v_2])$ is represented as an entry with a value 4 in row 2 and column 3 of the matrix in (3.5). Also, an edge with node u_i cannot be paired with any edge with v_i by **Prematch-III**. Thus, a value of an entry in row i and column i is 0.

The collection of all paired edges given by **Prematch-III** will be used as an input for an algorithm **Parallel-Edge-Pairing** which is given in Fig. 4. Consider in a matrix M, two entries e_k and e'_k in different columns such that $e_k = x'$ and e_k is in row y and column x, and $e'_k = x$ and e'_k is in row t and column x', $x \neq x'$. Then the entry e_k represents a pair of edges $p_1 = ([u_x, u_{x,x'}], [v_{y,y'}, v_y])$, and the entry e'_k represents a pair of edges $p_2 = ([u_{x'}, u_{x',x}], [v_{t,t'}, v_t])$. These entries e_k and e'_k in different columns of M represent pairs of edges having a common node $u_{x,x'} = u_{x',x}$. In this case, we say these entries are *common*. From Lemma D.1, the entry e_k in the matrix M has the unique common entry e'_k. We write $\overline{e_k}$ to indicate the entry e'_k. For example, in the matrix M in (3.5), let e_k $(= 2)$ be an entry in row 1 and column 4. Then $\overline{e_k}$ $(= 4)$ is an entry in row 3 and column 2. The algorithm **Parallel-Edge-Pairing** gives us a set E of $\min\{deg_f(u), deg_f(v)\}$ entries in M such that no two entries are common, and are in the same row or column. For each entry given by **Parallel-Edge-Pairing**, we can identify a corresponding pair of edges constructed by **Prematch-III**. We construct a path between them by **Procedure-III** whose sequence of labels in the path follows **Procedure-I**.

Procedure-III

Input: an entry e_k given by **Parallel-Edge-Pairing**.

Output: for a pair of edges $([u_i, u_{i,i'}], [v_{j,j'}, v_j])$ constructed by **Prematch-III** that corresponds with the given entry e_k, a fault-free path of the form $u_i\langle i', \ldots j'\rangle v_j$.

1. For the entry e_k given by **Parallel-Edge-Pairing**, identify the corresponding pair of edges constructed by **Prematch-III**.
2. If a corresponding pair of edges is of the form $([u_i, u_{i,i'}], [v_{i-1,i-2}, v_{i-1}])$, $1 \leq i, i' \leq r$ and $i' = i + 1$, then construct a path $u_i\langle i', \ldots i - 2\rangle v_{i-1}$ given by **Procedure-I**;

3. If a corresponding pair of edges is of the form $([u_i, u_{i,i'}], [v_{i'-1,i'-2}, v_{i'-1}])$, $i \leq i, i' \leq r$ and $i' = i-1$, then construct a path by flipping i' and i in the path $u_{i'}\langle i, \ldots, i'-2\rangle v_{i'-1}$ given by **Procedure-I**;

4. If a corresponding pair of edges is of the form $([u_i, u_{i,i'}], [v_{j,j'}, v_j])$, construct a path by flipping j and j' in the path $u_i\langle i', \ldots j\rangle v_{j'}$ given by **Procedure-I**;

Since the sequence in paths constructed by **Procedure-III** follows **Procedure-I**, flipping only the first two indices or the last two indices, Lemma C.1 can be applied to a path constructed by **Procedure-III** when the first two indices in the path are flipped. If, for a non-faulty pair of edges $([u_x, u_{x,x'}], [v_{y,y'}, v_y])$, a path P_x is constructed by flipping the last two indices, P_x shares some common nodes with a path P_s for a non-faulty paired edges $([u_s, u_{s,s'}], [v_{t,t'}, v_t])$ when either $u_{x,x'} = u_{s,s'}$, or $v_y = v_t$. Note that if $v_{y,y'}$ and $v_{t,t'}$ are identical, then $u_{x,x'}$ and $u_{s,s'}$ are also identical. Thus, Lemma C.1 can be applied to a path constructed by **Procedure-III** when the last two indices in the path are flipped. Now, when u and v have faulty neighbors, the problem of finding $\min\{deg_f(u), deg_f(v)\}$ non-faulty node-disjoint paths can be transformed into the problem of finding $\min\{deg_f(u), deg_f(v)\}$ entries in the matrix M such that no two entries are faulty and common, and are in the same row or column, plus constructing paths by **Procedure-III** for corresponding non-faulty node-disjoint pairs of edges given by **Prematch-III**.

Example: For a given two non-faulty nodes $u = 111100$ and $v = 000000$ in Q_n, let faulty nodes be $u_5 = 111110$, $v_6 = 000001$, $u_{2,3} = 100100$, and $u_{4,2} = 101000$. A matrix M of corresponding faulty entries (ones marked by $\otimes$ and $\times$) and entries e_1, e_2, e_3, e_4, and e_5 (ones in [ ]) such that no two entries are common and are in the same row or column is:

$$\begin{bmatrix} 0 & 3\otimes & 2\otimes & 2\otimes & 2\times & [e_5 = 2] \\ 3 & 0 & 4 & [e_4 = 3] & 3\times & 3 \\ [e_1 = 4] & 4\otimes & 0 & 1 & 4\times & 4 \\ 2 & [e_2 = 1] & 1 & 0 & 1\times & 1 \\ 5 & 5 & [e_3 = 5] & 5 & 0 & 5 \\ 6\times & 6\times & 6\times & 6\times & 6\times & 0 \end{bmatrix} \tag{3.6}$$

Node u_5 is faulty, so we mark all entries in column 5 as faulty (ones marked by $\times$). Similarly, node v_6 is faulty, so we mark all entries in row 6 as faulty. In this case, we say such a row or column is faulty. Also, node $u_{4,2}$ is faulty, so we mark the corresponding entry with the value 2 in row 1 and column 4 and the entry with the value 4 in row 3 and column 2 as faulty. Similarly, we mark faulty entries for a faulty node $u_{2,3}$. For disjoint non-faulty entries e_1, e_2, e_3, e_4, and e_5 in M, we identify pairs of edges constructed by **Prematch-III** as follows:

(1) for the entry e_1, the corresponding pair of edges is $([u_1, u_{1,4}], [v_{3,2}, v_3])$

(2) for the entry e_2, the corresponding pair of edges is $([u_2, u_{2,1}], [v_{4,3}, v_4])$

(3) for the entry e_3, the corresponding pair of edges is $([u_3, u_{3,5}], [v_{5,2}, v_5])$

(4) for the entry e_4, the corresponding pair of edges is $([u_4, u_{4,3}], [v_{2,1}, v_2])$

(5) for the entry e_5, the corresponding pair of edges is $([u_6, u_{6,2}], [v_{1,6}, v_1])$

For each corresponding pair of edges given by **Prematch-III**, we construct a path by **Procedure-III** as follows:

(1) for the paired edges $([u_1, u_{1,4}], [v_{3,2}, v_3])$, construct a path $u_1\langle 4, 2\rangle v_3$ by rule 3 of **Procedure 3** from the path $u_4\langle 1, 2\rangle e_3$ constructed by **Procedure-I**, flipping the first indices 4 and 1.

(2) for the paired edges $([u_2, u_{2,1}], [v_{4,3}, v_4])$, construct a path $u_2\langle 1, 3\rangle v_4$ by rule 3 of **Procedure 3** from the path $u_1\langle 2, 3\rangle e_4$ constructed by **Procedure-I**, flipping the first indices 1 and 2.

(3) for the paired edges $([u_3, u_{3,5}], [v_{5,2}, v_5])$, construct a path $u_3\langle 5, 4, 1, 2\rangle v_5$ by rule 4 of **Procedure 3** from the path $u_3\langle 5, 4, 1, 5\rangle e_2$ constructed by **Procedure-I**, flipping the last two indices 5 and 2.

(4) for the paired edges $([u_4, u_{4,3}], [v_{2,1}, v_2])$, construct a path $u_4\langle 3, 1\rangle v_2$ by rule 3 of **Procedure 3** from the path $u_3\langle 4, 1\rangle e_2$ constructed by **Procedure-I**, flipping the first two indices 3 and 4.

(5) for the paired edges $([u_6, u_{6,2}], [v_{1,6}, v_1])$, construct a path $u_6\langle 2, 3, 4, 6\rangle v_1$ by rule 4 of **Procedure 3** from the path $u_6\langle 2, 3, 4, 1\rangle e_6$ constructed by **Procedure-I**, flipping the last two indices 1 and 6.

Thus, the corresponding $\min\{deg_f(u), deg_f(v)\}$ node-disjoint paths from u to v derived from the matrix M in (3.6) are:

(1) $u\langle 1, 4, 2, 3\rangle v$

(2) $u\langle 2, 1, 3, 4\rangle v$

(3) $u\langle 3, 5, 4, 1, 2, 5\rangle v$

(4) $u\langle 4, 3, 1, 2\rangle v$

(5) $u\langle 6, 2, 3, 4, 6, 1\rangle v$

In Figure 4, we present the algorithm **Parallel-Edge-Pairing** that finds $\min\{deg_f(u), deg_f(v)\}$ non-faulty entries such that no two entries are common, and are in the same row or column of the matrix M.

Definition For a row R or a column C of the matrix M, the row R or the column C is said to be *feasible* if in R or C, there is a non-faulty entry whose value is not 0. Also, for an entry e_i in a row R and a column C of the matrix M, and a set E of non-faulty entries such that no two entries are common, and are in the same row or column in M, the row R, the column C, and the entry $\overline{e_i}$ are said to be *used* by e_i if the entry e_i is in E.

From the above definition, if for an entry e_i in a row R and a column C, the entry e_i is in E, then R or C is used by the entry e_i, and the entry $\overline{e_i}$ is also used. If in R or C, there is no entry in E, then R or C is unused. Thus, for an entry $\overline{e_i}$ in a row R' and a column C', R' or C' is unused unless R is R', C is C', or an entry e_j, $i \neq j$ in R' or C' is in E. In addition, a row or column used by the entry e_i in E is feasible because the value of e_i in E is not 0, and e_i is not faulty. We assume that when an entry e_i in a row R and a column C is added to E or removed from E, the entry $\overline{e_i}$, the row R, and the column C are accordingly marked as used or unused.

Let us denote notations for describing algorithms. For an entry e_i of M, we denote $R(e_i)$ as a row with e_i, and $C(e_i)$ as a column with e_i. Let R be a row j of M. Then entries in R of M represent pairs of edges with a node v_j. If $1 \leq j \leq r$, then the row R consists of n entries such that one entry has a value 0, one entry has a value $j+2$, and $n-2$ entries have a value $j+1$. Also, if $r+1 \leq j \leq n$, then the row R consists of n entries such that one entry has a value 0, and $n-1$ entries have a value j. Let C be a column j of M. Then we denote $\overline{C}$ as a row that represents pairs of edges with a node v_{j-1} if $1 \leq j \leq r$, or with a node v_j if $r+1 \leq j \leq n$. Similarly, for a row R that represents pairs of edges with a node v_{j-1} if $1 \leq j \leq r$, or with a node v_j if $r+1 \leq j \leq n$, we denote $\overline{R}$ as a column j. Then for an entry e_i in R, $\overline{e_i}$ is in $\overline{R}$. For example, in the matrix M in (3.5), row 3 consists of entries such that one entry has a value 0, one entry has a value 1, and 4 entries have a value 4. Also, row 6 consists of entries such that one entry has a value 0 and 5 entries have a value 6. Let R be row 1, and C be column 6. Then $\overline{R}$ is column 2, and $\overline{C}$ is row 6. Also, for an entry $e_i = 2$ in row 1 (which is R) and column 5, $\overline{e_i} = 5$ is in row 5 and column 2 (which is $\overline{R}$).

Let α be the set of faulty rows in M, and β be the set of faulty columns in M. Then we denote $R(e_i)/\beta$ as a set of entries in $R(e_i)$ excluding entries in faulty columns, and $C(e_i)/\alpha$ as a set of entries in $C(e_i)$ excluding entries in faulty rows. For two entries e_i and e_j in M, we write as $e_i \rightarrow e_j$ if e_i and e_j are in the same row, and e_i is visited before e_j. Also, we write as $e_i \Rightarrow e_j$ if e_i and e_j are in the same column, and e_i is visited before e_j. Finally, we write as $e_i = e_j$, if e_i and e_j are identical such that e_i and e_j have the same value and are in the same row and column. Otherwise, we write $e_i \neq e_j$. In Fig. 4, we present the algorithm **Parallel-Edge-Pairing** that finds $\min\{deg_f(u), deg_f(v)\}$ non-faulty entries in M.

For a set E of Δ entries in M such that no two entries are common and are in the same row or column, the algorithm **Augmenting** is used to find the set E of size $\Delta + 1$, or of size $\min\{deg_f(u), deg_f(v)\}$ when all non-faulty entries in an unused column are used, or are in used rows. The algorithm **Augmenting** is given in Fig. 5. Specifically, in the algorithm **Augmenting**, we use the algorithm **Augmenting-I** to handle the case that there is a used entry in unused row and column, and use the algorithm **Augmenting-II** to handle the case that all non-faulty entries in an unused column are in used rows. The algorithm **Augmenting-I** is given in Fig. 6.

To show the correctness of the algorithm **Augmenting-I**, we first discuss the correctness of the algorithms **Aug-I** given in Fig. 7 and **Aug-II** given in Fig. 8

Algorithm. Parallel-Edge-Pairing

Input: the matrix M of entries that correspond with edges paired by **Prematch-III**.

Output: a set E of $\min\{deg_f(u), deg_f(v)\}$ non-faulty entries in M such that no two entries are common and are in the same row or column. Initially, $E = \phi$.

1. find a non-faulty and unused entry e_i;
2. $E = \{e_i\}$;
3. **for** each feasible and unused column C **do**

3.1 **if** $|E|=\min\{deg_f(u), deg_f(v)\}$ **then** STOP.

3.2 find a feasible and unused row R;

3.3 **if** there is an entry e_j in C such that e_j is non-faulty, $\overline{e_j}$ is not in E, and $R(e_j)$ is unused
then $E = E \cup \{e_j\}$;

3.4 **else if** all non-faulty entries in C are used, or are in used rows
  if $\overline{R}$ is C
  then find a non-faulty entry e_j in C such that $e_j \rightarrow e'_j$
    and e'_j is in E;
    $E = E - \{e'_j\}$; $E = E \cup \{e_j\}$;
    let $C(e'_j)$ be C;
    go to step 3.3;
  else call Augmenting(E, R, C);

Fig. 4. Parallel edge-pairing on the hypercube network with faulty nodes when both u and v have faulty neighbors

by going through each step, assuring that entries in E are not common and are in different rows and columns. For a used entry e_1 in feasible and unused R and C such that an entry $\overline{e_1}$ is in E, let C be a column i, b_1 be an entry in $R(e_1)$ and $C(\overline{e_1})$, and b_2 be an entry in $R(\overline{e_1})$ and $C(e_1)$.

Lemma D.2 *For a given set E of entries such that no two entries are common and are in the same row or column, feasible and unused row R and column C, and a used entry e_1 in R and C, if the entry $\overline{e_1}$ is the only non-faulty entry in $R(\overline{e_1})$, then the algorithm* **Aug-I** *increases the size of E by one.*

Algorithm. Augmenting(E, R, C**)**

Input: a set E of Δ entries such that no two entries are common, and are in the same row or column, feasible and unused row R and column C.

Output: the set E of size $\Delta + 1$, or of size $\min\{deg_f(u), deg_f(v)\}$.

1. **case 1.** for an entry e_1 in R and C, e_1 is used by $\overline{e_1}$
   call Augmenting-I$(E, R(e_1), C(e_1), e_1)$;
2. **case 2.** in C, all non-faulty entries $e_1, e_2, \ldots, e_h$ are in rows used by $e'_1, e'_2, \ldots, e'_h$ such that for all i, $1 \le i \le h$, e'_i is in E, and $e_i \rightarrow e'_i$, respectively
   call Augmenting-II(E, R, C);

Fig. 5. The algorithm **Augmenting**

Algorithm. Augmenting-I($E, R(e_1), C(e_1), e_1$**)**

Input: a set E of Δ entries such that no two entries are common and are in the same row or column, feasible and unused row $R(e_1)$ and column $C(e_1)$, and a used entry e_1 in $R(e_1)$ and $C(e_1)$ such that $\overline{e_1}$ is in E.

Output: the set E of size $\Delta + 1$, or of size $\min\{deg_f(u), deg_f(v)\}$.

1. Let $C(e_1)$ be a column i, b_1 be an entry in $R(e_1)$ and $C(\overline{e_1})$, and b_2 be an entry in $R(\overline{e_1})$ and $C(e_1)$;
2. **case 1.** $\overline{e_1}$ is the only non-faulty entry in $R(\overline{e_1})$
   call Aug-I$(E, R(e_1), C(e_1), e_1)$;
3. **case 2.** both entries b_1 and b_2 are non-faulty
   $E = E - \{\overline{e_1}\}$; $E = E \cup \{b_1, b_2\}$;
4. **case 3.** b_1 is non-faulty but b_2 is 0 or faulty

4.1 **if** there is a non-faulty entry k_3 in $R(e_1)$ such that $k_1 \rightarrow k_2$ and $k_2 \Rightarrow k_3$ where k_1 is non-faulty and is in $C(e_1)$, and k_2 is in E
   then $E = E - \{k_2\}$; $E = E \cup \{k_1, k_3\}$;

4.2 **else** call Aug-II$(E, R(e_1), C(e_1), e_1)$;

5. **case 4.** b_1 is 0 or faulty

5.1 **if** b_2 is non-faulty
   then $E = E - \{\overline{e_1}\}$; $E = E \cup \{e_1\}$;
   call Augmenting-I$(E, R(\overline{e_1}), C(\overline{e_1}), \overline{e_1})$;

5.2 **else** use Aug-II$(E, R(e_1), C(e_1), e_1)$, where for step 3.3 in Aug-II, apply the algorithm used in case 3 of Aug-II;

Fig. 6. The algorithm **Augmenting-I**

Algorithm. Aug-I($E, R(e_1), C(e_1), e_1$**)**

Input: a set E of Δ entries in M such that no two entries are common and are in the same row or column, feasible and unused row $R(e_1)$ and column $C(e_1)$, and a used entry e_1 in $R(e_1)$ and $C(e_1)$ such that $\overline{e_1}$ is in E.

Output: the set E of size $\Delta + 1$.

{Assumption: $\overline{e_1}$ is the only non-faulty entry in $R(\overline{e_1})$.}

1. find a non-faulty entry e_2 in $C(e_1)$, where $e_2 = i - 1$, $1 \leq i \leq r$;
2. find an entry e_4 in $R(e_1)$ such that $e_2 \rightarrow e_3$ and $e_3 \Rightarrow e_4$ where e_3 is in E;
3. **if** e_4 is non-faulty
   then $E = E - \{e_3\}$; $E = E \cup \{e_2, e_4\}$;
4. **else** $e_4 = 0$
   find a non-faulty entry e_5 such that $e_3 \Rightarrow e_5$, and e_5 and $\overline{e_5}$ are in the same row, or e_5 is unused;
   if $R(e_5)$ is unused
   then $E = E - \{e_3\}$; $E = E \cup \{e_2, e_5\}$;
   else ($R(e_5)$ is used by e_6 in E)
   find a non-faulty entry e_7 in $R(e_1)$ such that $e_5 \rightarrow e_6$ and $e_6 \Rightarrow e_7$;
   $E = E - \{e_3, e_6\}$; $E = E \cup \{e_2, e_5, e_7\}$;

Fig. 7. The algorithm **Aug-I**

PROOF. Suppose, in a column i, C, there is a used entry $e_1(= j)$ by $\overline{e_1}$ such that $\overline{e_1}$ is in E. If $j = i - 1$ or $i + 1$, $1 \leq i, j \leq r$, then both e_1 and $\overline{e_1}$ are in the same row $R(e_1)$. Since we assume that e_1 is used, and $R(e_1)$ is unused such that $R(e_1)$ does not contain an entry in E, we have $j \neq i - 1$ or $i + 1$ when $1 \leq i, j \leq r$. Suppose $\overline{e_1}$ is the only non-faulty entry in $R(\overline{e_1})$. Then i is in between 1 and r, and there must be at least $n - 3$ faulty entries in $R(\overline{e_1})$, and thus $|\alpha| = 1$. Since $|\alpha| = 1$ and Q_n has at most $n - 2$ faulty nodes, $R(e_1)/\beta$ does not have faulty entries. If $n = 4$, then we have one faulty row and one faulty column. Thus, in $R(\overline{e_1})$, there exists another non-faulty entry except $\overline{e_1}$. Therefore, we assume that $n \geq 5$.

Step 1 of the algorithm

Since the value of an entry in $R(\overline{e_1})$ is 0, i, or $i + 1$ (that is, $R(\overline{e_1})$ is $\overline{C(e_1)}$), all entries in C except entries e_1 and e_2 $(= i - 1)$ are faulty. Note that e_2 and $\overline{e_2}$ are in

Algorithm. Aug-II$(E, R(e_1), C(e_1), e_1)$

Input: a set E of Δ entries such that no two entries are common, and are in the same row or column, feasible and unused row $R(e_1)$ and column $C(e_1)$, and a used entry e_1 in $R(e_1)$ and $C(e_1)$ such that $\overline{e_1}$ is in E.

Output: the set E of size $\Delta + 1$, or of size $\min\{deg_f(u), deg_f(v)\}$.

{Assumption: the entry b_1 in $R(e_1)$ and $C(\overline{e_1})$ is non-faulty, but the entry b_2 in $R(\overline{e_1})$ and $C(e_1)$ is 0 or faulty. Also, there is no non-faulty unused entry k_3 such that $k_1 \rightarrow k_2$ and $k_2 \Rightarrow k_3$ where k_1 is non-faulty and is in $C(e_1)$, and k_2 is in E. }

1. find a non-faulty entry $e_2(\neq \overline{e_1})$ in $R(\overline{e_1})$;
2. **if** $C(e_2)$ is unused
   then $E = E - \{\overline{e_1}\}$; $E = E \cup \{e_1, e_2\}$;
3. **else**($C(e_2)$ is used by e_3 in E)

3.1 find an entry e_4 in $R(e_3)$ and $C(\overline{e_1})$ such that $e_3 \rightarrow e_4$ and e_3 is in E;

3.2 **if** e_4 is non-faulty
    then $E = E - \{\overline{e_1}, e_3\}$; $E = E \cup \{e_1, e_2, e_4\}$;

3.3 **else** $e_4 = 0$ or faulty
    find an entry e_5 in $R(e_3)$ and $C(e_1)$ such that $e_3 \rightarrow e_5$,
    case 1. e_5 is non-faulty, and e_5 and e_2 are not common
        $E = E - \{\overline{e_1}, e_3\}$; $E = E \cup \{b_1, e_2, e_5\}$;
    case 2. e_5 and e_2 are common and $e_4 = 0$
        $E = E - \{e_3\}$; $E = E \cup \{\overline{b_1}, e_5\}$;
    case 3. e_5 and e_2 are common and e_4 is faulty, or e_5 is 0 or faulty
        find a non-faulty entry e_6 such that $e_3 \rightarrow e_6$;
        if $C(e_6)$ is unused
            if e_6 is unused, or e_6 and e_3 are common
            then $E = E - \{\overline{e_1}, e_3\}$; $E = E \cup \{e_1, e_2, e_6\}$
            else call BFS(M);
        if $C(e_6)$ is used by e_7 in E
            find a non-faulty entry e_8 in $C(e_1)$ or $C(\overline{e_1})$ such that $e_3 \rightarrow e_6$, $e_6 \Rightarrow e_7$, and $e_7 \rightarrow e_8$, where e_8 and e_2 are not common;
            if e_8 is unused, or e_8 and e_7 are common
            then $E = E - \{\overline{e_1}, e_3, e_7\}$; $E = E \cup \{e_1, e_2, e_6, e_8\}$
            else call BFS(M);

Fig. 8. The algorithm **Aug-II**

the same row.

Step 2 of the algorithm

In step 3.4 of the algorithm **Parallel-Edge-Pairing**, we assume that all non-faulty entries in column i are used or are in used rows. Thus, e_2 is used by an entry e_3, or is in $R(e_2)$ used by an entry e_3 in E such that $e_2 \rightarrow e_3$. Let e_4 be an entry in $R(e_1)$ such that $e_3 \Rightarrow e_4$.

Step 3 of the algorithm

Suppose e_4 is non-faulty. Then e_4 must be unused because either $\overline{e_4}$ is in $R(e_1)$ or in $C(\overline{e_1})$. Thus, e_4 can be added to E after removing e_3 from E. Next, we want to insert an entry e_2 to E. Since $e_2 = i - 1$, $1 \leq i \leq r$, and e_2 and $\overline{e_2}$ are in the same row, entries e_2 and e_4 cannot be common. Also, the entry e_2 is unused unless $\overline{e_2}$ is e_3. The entry e_4 is in $R(e_1)$, so $R(e_2)$ and $R(e_4)$ are different rows. Also, the entry e_4 is in $C(e_4)$ which is used by e_3 in E, so $C(e_2)$ and $C(e_4)$ are different columns. Thus, e_2 can be added to E. It shows that the size of E increases by one because we remove e_3 from E, and add e_2 and e_4 to E.

Step 4 of the algorithm

If $e_4 = 0$, then e_4 cannot be added to E, and we need to search other entries. Note that e_4 cannot be faulty because $R(e_1)/\beta$ does not have faulty entries. Let $C(e_3)$ be a column a, $a \neq i$. Then $C(e_3)/\alpha$ does not have faulty entries except an entry e' in $C(e_3)$ and $R(\overline{e_1})$ because common entries of all faulty entries in $R(\overline{e_1})$ are in $C(e_1)$. We show that we can find a non-faulty entry e_5 ($\neq e_4$) in $C(e_3)$ such that e_5 and $\overline{e_5}$ are in the same row, or e_5 is unused. Suppose $n = 5$, then there is at least one non-faulty entry e_5 such that $e_3 \Rightarrow e_5$ because $|\alpha| = 1$. If e_5 is used by $\overline{e_5}$ in E, then e_5 and $\overline{e_5}$ must be in the same row because one row must be faulty, and other rows $R(\overline{e_1})$ and $R(e_2)$ are used by $\overline{e_1}$ and e_3, respectively. Suppose $n \geq 6$, then there exists at least two non-faulty entries in $C(e_3)$ except e_3. Let these entries be b and

b'. If b (or b') is $a-1$ or $a+1$ where $1 \leq a, b, b' \leq r$, then we can find an entry e_5 such that e_5 and $\overline{e_5}$ are in the same row. Otherwise, entries b and b' cannot be used at the same time because their common entries $\bar{b}$ and $\overline{b'}$ would be in the same row. In that case, one must be unused.

Now, if $R(e_5)$ is unused, then e_5 can be added to E after removing e_3 from E. Since $e_2 = i-1$ and is in $C(e_1)$, e_2 and e_5 are not common. Also, if e_2 is used, then $\overline{e_2}$ is e_3. Since $R(e_2)$ and $R(e_5)$ are different, and $C(e_2)$ and $C(e_5)$ are also different, e_2 can be added to E, and the size of E increases by one. If $R(e_5)$ is used by an entry e_6 in E, then we need to show that there exists a non-faulty unused entry e_7 in $R(e_1)$ such that $e_5 \rightarrow e_6$ and $e_6 \Rightarrow e_7$. A value of e_7 is j or $j+1$. Note that the value of e_7 cannot be 0 because $e_4 = 0$. If $e_7 = j$, then $\overline{e_7}$ is in $C(\overline{e_1})$, which is already used by $\overline{e_1}$ in E. If $e_7 = j+1$, then e_7 is unused because $\overline{e_7}$ is also in the unused row $R(e_1)$, and thus, $\overline{e_7}$ cannot be in E. The entry e_5 is not in $R(e_1)$, so $R(e_5)$ and $R(e_7)$ are different. Also, the entry e_5 is not in $C(e_6)$ and $e_6 \Rightarrow e_7$, so $C(e_5)$ and $C(e_7)$ are different. Now, e_5 and e_7 can be added to E after removing e_3 and e_6 from E. Similarly, we can show entries e_2, e_5, and e_7 are not common, and $R(e_2)$ is different to $R(e_5)$ or $R(e_7)$, and $C(e_2)$ is different to $C(e_5)$ and $C(e_7)$. Therefore, e_2 can be added to E, and the size of E increases by one.

From the above discussion, we show that for the given set E of entries in M such that no two entries are common and are in the same row or column, feasible and unused row and column with the used entry in the algorithm **Aug-I**, the algorithm **Aug-I** increases the size of E by one. $\square$

Lemma D.3 *For a given set E of entries such that no two entries are common, and are not in the same row of column, feasible and unused row R and column C, and a used entry e_1 in R and C, if the entry b_1 in $R(e_1)$ and $C(\overline{e_1})$ is non-faulty, but the* *entry b_2 in $R(\overline{e_1})$ and $C(e_1)$ is 0 or faulty, and there is no non-faulty unused entry k_3 such that $k_1 \rightarrow k_2$ and $k_2 \Rightarrow k_3$ where k_1 is non-faulty and is in $C(e_1)$, and k_2 is in E, then the algorithm* **Aug-II** *increases the size of E by one, or finds the set E of size $min\{deg_f(u), deg_f(v)\}$.*

Proof. Let b_1 be an entry in $R(e_1)$ and $C(\overline{e_1})$, and b_2 be an entry in $R(\overline{e_1})$ and $C(e_1)$.

Step 1 of the algorithm

Suppose there are non-faulty entries b_1 and $e_2(\neq b_2)$ in $R(\overline{e_1})$. First, we show that e_2 must not be used by $\overline{e_2}$ in E. The value of an entry e_2 is i, and then $\overline{e_2}$ is in $C(e_1)$. Since $C(e_1)$ is unused, $\overline{e_2}$ cannot be in E. Thus, e_2 is unused.

Step 2 of the algorithm

Suppose $C(e_2)$ is unused. Then we can add e_2 to E after removing $\overline{e_1}$ from E. Since $\overline{e_1} \neq e_2$, e_1 and e_2 are not common. Also, $R(e_1)$ and $R(e_2)$ are different because we assume that $R(e_1)$ is unused. Since $C(e_1)$ and $C(e_2)$ are different, and e_1 becomes unused by removing $\overline{e_1}$ from E, we can add e_1 to E, and the size of E increases by one.

Step 3 of the algorithm

Suppose $C(e_2)$ is used by an entry e_3 in E.

Step 3.1 of the algorithm

If $C(e_2)$ is used by an entry e_3 in E, then we find an entry e_4 in $R(e_3)$ and $C(\overline{e_1})$ such that $e_3 \rightarrow e_4$ and e_3 is in E. Note that $e_4 \neq b_1$ because e_3 cannot be in $R(e_1)$.

Step 3.2 of the algorithm

Suppose e_4 is non-faulty. If $e_4 = j-1$, $1 \leq j \leq r$, then e_4 and $\overline{e_4}$ are in the same row. In this case, $\overline{e_4}$ must be e_3 if e_4 is used. Otherwise, e_4 is unused because $\overline{e_4}$ is in $R(e_1)$. Also, e_4 is not $\overline{e_2}$. Thus, we can add e_4 to E after removing $\overline{e_1}$, and e_3. We assume $e_2 \Rightarrow e_3$ and $e_3 \rightarrow e_4$, so e_2 and e_4 are not in the same row or column. Also, by removing $\overline{e_1}$ and e_3 from E, $R(e_2)$ and $C(e_2)$ become unused. Thus, we can add e_2 to E. Finally, we want to add e_1 to E. An entry $\overline{e_1}$ is not e_2 or e_4. Thus, e_1, e_2, and e_4 are not common. Since $R(e_1)$ or $C(e_1)$ cannot contain e_3 or $\overline{e_1}$, $R(e_1)$ is different to $R(e_2)$ or $R(e_4)$, and $C(e_1)$ is different to $C(e_2)$ or $C(e_4)$. Therefore, we can add e_1 to E, and the size of E increases by one because we remove $\overline{e_1}$ and e_3 from E, and add e_1, e_2, and e_4.

Step 3.3 of the algorithm

Suppose $e_4 = 0$ or e_4 is faulty. Then we find an entry e_5 in $R(e_3)$ and $C(e_1)$ such that $e_2 \Rightarrow e_3$ and $e_3 \rightarrow e_5$ where e_3 is in E.

Case 1. e_5 is non-faulty, and e_5 and e_2 are not common.

If e_5 and e_2 are not common, then an entry $\overline{e_5}$ is in $R(\overline{e_1})$, or e_5 and $\overline{e_5}$ are in the same row $R(e_3)$. Thus, we can add e_2 and e_5 to E by removing $\overline{e_1}$ and e_3. Now we can add $b_1 = j + 1$ in $R(e_1)$ and $C(\overline{e_1})$ to E, and the size of E increases by one because we remove $\overline{e_1}$ and e_3 from E, and add b_1, e_2, and e_5 to E.

Case 2. e_5 and e_2 are common and $e_4 = 0$.

Let k be a value of e_5. Note that k must be between 1 and r because $\overline{R(e_2)}$ is $C(e_5)$, and e_3 is non-faulty. Then a value of e_3 must be $k+1$, and e_3 is in row $k-1$ and column k because e_5 and e_2 are common. Also, the value of an entry in row $k-1$ and column $k-1$ is 0. Since we assume that $e_4 = 0$ and e_4 is in $R(e_3)$ and $C(\overline{e_1})$, the entry e_4 is in row $k-1$ and column $k-1$. From this observation, we know that $C(\overline{e_1})$ is column $k-1$. Also, b_1 $(= k)$ is in row $k-2$ and column $k-1$, and $\overline{b_1}$ $(= k-1)$ is in row $k-2$ and column k. Thus, we can add $\overline{b_1}$ and e_5 to E after removing e_3 from E. Obviously, $\overline{b_1}$ and e_5 are not common, and they are in different rows and columns. Thus, the size of E increases by one because we remove e_3 from E and add

$\overline{b_1}$ and e_5.

Case 3. e_5 and e_2 are common and e_4 is faulty, or $e_5 = 0$ or faulty.

Let p be the number of non-faulty unused entries except e_5 in $R(e_3)$, including $\overline{e_3}$ when it is non-faulty and is in $R(e_3)$. Also, let q be the number of faulty entries in $R(\overline{e_1})/\beta$, and q' be the number of faulty entries in $C(e_1)/\alpha$.

Case 3.1. e_5 and e_2 are common and e_4 is faulty.

First, we search a non-faulty entry e_6 such that $e_3 \rightarrow e_6$, and e_3 is in E. We show that $p \geq 1$. For entries b_1 and e_4 where $e_2 \Rightarrow e_3$ and $e_3 \rightarrow e_4$, we assume that b_1 is non-faulty, and e_4 is faulty. It implies that there are at least $(q+|\beta|)+(n-q-|\beta|-3) = n-3$ faulty entries in $R(\overline{e_1})$ and $C(\overline{e_1})/\alpha$. Thus, we have $|\alpha| = 1$. Since we already found at least $n-2$ faulty entries in M, there is no faulty entry in $R(e_3)/\beta$ except e_4. Also, there is no used entry in $R(e_3)$ except $\overline{e_3}$ because e_5 and e_2 are common. It implies $R(e_3)$ is $\overline{C(e_3)}$, and $R(e_3)$ and $C(e_3)$ are only used by e_3. Thus, $p \geq n-|\beta|-4$ because we exclude an entry with a value 0, e_3, e_4, and e_5, where $1 \leq |\beta| \leq n-3-4 = n-7$. Thus, unless $n \leq 5$ and $|\beta| = 1$, we have $p \geq 1$, which implies we can find at least one non-faulty unused entry e_6 in $R(e_3)$, or a used entry e_6 such that e_6 and e_3 are common. Suppose $n \leq 5$ and $|\beta| = 1$. If $b_2 \neq 0$, then b_2 and e_4 must be faulty. Since $|\alpha| \geq 1$ and $\beta| \geq 1$, it is easy to prove that there would be more than $n-2$ faulty nodes in Q_n. If $b_2 = 0$, then the number of faulty entries in $R(\overline{e_1})$ and $C(\overline{e_1})/\alpha$ is at least $(q+|\beta|)+(n-q-|\beta|-2) = n-2$. Since $|\alpha| \geq 1$, there are at least $n-1$ faulty nodes in Q_n, which again contradicts the assumption that there are at most $n-2$ faulty nodes in Q_n. Therefore, we can find at least one non-faulty unused entry e_6 or a used entry e_6 (where e_6 and e_3 are common) in $R(e_3)$. Now, if $C(e_6)$ is unused, then we can add e_1, e_2 and e_6 to E after removing $\overline{e_1}$ and e_3 from E.

If $C(e_6)$ is used by an entry e_7 in E, then we continuously search a non-faulty unused entry e_8 in $C(e_1)$ or $C(\overline{e_1})$ such that $e_3 \rightarrow e_6$, $e_6 \Rightarrow e_7$, and $e_7 \rightarrow e_8$, where e_8 and e_2 are not common. We show that there exists at least one such non-faulty unused entry e_8 in $C(e_1)$ or $C(\overline{e_1})$. Consider p non-faulty entries d_j, $1 \leq j \leq p$ in $R(e_3)$ such that d_j is unused or $\overline{e_3}$, and $d_j \Rightarrow d'_j$, $d'_j \rightarrow d''_j$, and $d'_j \rightarrow d'''_j$ where d'_j is in E, d''_j is in $C(\overline{e_1})$, and d'''_j is in $C(e_1)$. Assume by way of contradiction that entries d''_j and d'''_j, $1 \leq j \leq p$, are all faulty or 0. Let q'' be the number of faulty entries in $R(\overline{e_1})$ plus the number of entries d''_j , $1 \leq j \leq p$, which are faulty or 0. Then $q'' = |\beta| + q + p \leq n - 3$, assuming that there is an entry d''_j such that $d''_j = 0$. Since $p \geq 1$, d'''_j in $C(e_1)$ is faulty where $d'_j \rightarrow d'''_j$, and thus $\overline{d'''_j}$ is in $R(\overline{e_1})$. We also showed that the value of b_2 cannot be 0. Thus, b_2 $(= i + 1)$ and $\overline{d'''_j}$ are faulty, and $q \geq 2$. Therefore, we have $p \leq n - |\beta| - 5$, contradicting the assumption that $p \geq n - |\beta| - 4$. It shows that there exists at least one non-faulty entry d''_j in $C(\overline{e_1})$. It contradicts the assumption that entries d''_j, $1 \leq j \leq q$, are all faulty or 0. If $\overline{d''_j}$ is in $R(e_1)$ or $R(d'_j)$, then there exists at least one non-faulty entry e_8 in $C(\overline{e_1})$ such that e_8 is unused, or e_8 and e_7 are common. Since e_8 in $C(\overline{e_1})$ and e_2 in $R(\overline{e_1})$ cannot be common, we can add e_1, e_2, e_6, and e_8 to E after removing $\overline{e_1}$, e_3, and e_7.

Case 3.2. $e_5 = 0$ or e_5 is faulty.

Similar to case 3.1, first we search a non-faulty entry e_6 such that $e_3 \rightarrow e_6$ and e_3 is in E. We show that for most of cases, $p \geq 1$ holds. In case $p < 1$, we show n is constant and provide $\min\{deg_f(u), deg_f(v)\}$ non-faulty entries in M by using the algorithm BFS(M) which is shown in Fig. 9. If $C(e_6)$ is used by e_7, then we continuously search a non-faulty entry e_8 in $C(e_1)$ or $C(\overline{e_1})$ such that e_8 and e_2 are not common. To prove the existence of such an entry e_8, we again consider p non-faulty entries d_j, $1 \leq j \leq p$ in $R(e_3)$ such that d_j is unused or $\overline{e_3}$ and $d_i \Rightarrow d'_j$, $d'_j \rightarrow d''_j$, and $d'_j \rightarrow d'''_j$ where d'_j is in E, d''_j is in $C(\overline{e_1})$, and d'''_j is in $C(e_1)$. By way of contradiction, we assume again that entries d''_j and d'''_j, $1 \leq j \leq p$, are all faulty or 0.

Algorithm. BFS(M)

Input: the matrix M of entries that correspond with edges paired by **Prematch-III**

Output: a set E of $\min\{deg_f(u), deg_f(v)\}$ non-faulty entries in M such that no two entries are common, and are in the same row or column. Initially, $E = \phi$.

Phase 1: construct a network $G = (X, Y)$ from a source node s, where X is a set of nodes and Y is a set of edges in G.

{Assumption: each node in G contains the field *row*, *column*, and *set*. }

let Q be a queue. Initially, $Q = \phi$;
$X = \{s\}$; $Y = \phi$;
$row[s]$ = NULL; $column[s]$ = NULL; $set[s]$ = NULL;
$Q \leftarrow s$;
while Q is not empty **do**
  $v \leftarrow Q$;
  for each non-faulty entry e in an each column C of M **do**
    if $\overline{e}$ is not in $set[v]$, $R(e)$ is not in $row[v]$, and $C(e)$ is not in $column[v]$
    then $Q \leftarrow e$;
      $X = X \cup \{e\}$; $Y = Y \cup \{[v, e]\}$;
      $row[e] = row[v] \cup R(e)$;
      $column[e] = column[v] \cup C(e)$;
      $set[e] = set[v] \cup \{e\}$;

Phase 2: find a node e in G such that $|set[e]|$=$\min\{deg_f(u), deg_f(v)\}$.

perform a Breath first search on G starting from s;
if for a node e in G, the size of $set[e]$ is $\min\{deg_f(u), deg_f(v)\}$, then
let $E = set[e]$, and STOP.

Fig. 9. The algorithm **BFS**

We show that most cases find at least one non-faulty entry d''_j in $C(\overline{e_1})$. If we cannot find such an entry d''_j, we again show n is constant and provide $\min\{deg_f(u), deg_f(v)\}$ non-faulty entries in M by using the algorithm BFS(M). We discuss each case in detail.

Case 3.2.1. $e_5 = 0$ and e_4 is faulty.

Since there are at least $(q + |\beta|) + (n - q - |\beta| - 3) = n - 3$ faulty entries in $R(\overline{e_1})$ and $C(\overline{e_1})/\alpha$, there is no faulty entry in $R(e_3)/\beta$ except e_4. If $e_4 = j - 1$, then $p \geq n - |\beta| - 5$ because we exclude e_3, e_4, $\overline{e_4}$, e_5, and a used entry e' such that e' and e_3 are not common, where $1 \leq |\beta| \leq n - 3 - 5 = n - 8$. If $e_4 \neq j - 1$, then

$p \geq n - |\beta| - 4$ because we exclude e_3, e_4, e_5, and a used entry e' such that e' and e_3 are not common, where $1 \leq |\beta| \leq n - 3 - 4 = n - 7$. Thus, unless $n \leq 6$ and $|\beta| = 1$ when $e_4 = j - 1$, or unless $n \leq 5$ and $|\beta| = 1$ when $e_4 \neq j - 1$, there exists at least one non-faulty unused entry e_6 or a used entry e_6 (where e_6 and e_3 are common) in $R(e_3)$. In this case, if $C(e_6)$ is unused, then we can add e_1, e_2, and e_6 to E after removing $\overline{e_1}$ and e_3 from E. If $n \leq 5$, then b_2 $(= i+1)$ and e_4 are faulty. Since $|\alpha| \geq 1$ and $|\beta| \geq 1$, there are at least four faulty nodes in Q_n, $n \leq 5$. It contradicts the assumption that there are at most $n - 2$ faulty nodes in Q_n. Consider the case $e_4 = j - 1$ when $n = 6$ and $|\beta| = 1$. In this case, it is possible that there is only one non-faulty entry e_6 which is used by $\overline{e_6}$ $(\neq e_3)$. Since there is only one faulty entry b_2 in $C(e_1)$, there are at least two non-faulty entries in $C(e_1)$ except e_1. Thus, non-faulty entries k_1, k_2, and k_3 exist such that $k_1 \rightarrow k_2$ and $k_2 \Rightarrow k_3$ where k_1 is in $C(e_1)$, k_2 is in E, and k_3 is in $R(e_1)$ which falls into step 4.1 of the algorithm **Augmenting-I**.

If $C(e_6)$ is used by e_7 in E, then similar to case 3.1, we continuously search a non-faulty entry e_8 in $C(e_1)$ or $C(\overline{e_1})$. Suppose $n \geq 7$ and $|\beta| \geq 1$ when $e_4 = j - 1$. Then we showed that $p \geq n - |\beta| - 5 \geq 1$. Since b_2 $(= i + 1)$ is faulty, we have $q \geq 1$. In this case, $q'' = |\beta| + p + q \leq n - 3$. Since $q \geq 1$, we have $p \leq n - |\beta| - 4$, which implies all entries d''_j, $1 \leq j \leq p$, in $C(\overline{e_1})$ can be faulty or 0. However, in this case, we claim non-faulty entries k_1, k_2, and k_3 exist such that $k_1 \rightarrow k_2$ and $k_2 \Rightarrow k_3$ where k_1 is in $C(e_1)$, k_2 is in E, and k_3 is in $R(e_1)$ which falls into step 4.1 of the algorithm **Augmenting-I**. For $h \geq n - q' - |\alpha| - 2$ non-faulty entries in $C(e_1)$ except e_1, consider entries k_j, k'_j, and k''_j, $1 \leq j \leq h$ such that $k_j \rightarrow k'_j$ and $k'_j \Rightarrow k''_j$ where k_j is in $C(e_1)$, k'_j is in E, and k''_j is in $R(e_1)$. Recall that q' is the number of faulty entries in $C(e_1)/\alpha$. If all k''_j, $1 \leq j \leq h$, are faulty or 0, then the number of faulty entries in $C(e_1)$ plus such entries k''_j, $1 \leq j \leq h$ is $q' + |\alpha| + h$, which is bounded by $n - 3$. Thus, $h \leq n - q' - |\alpha| - 3$, contradicting the assumption that $h \geq n - q' - |\alpha| - 2$.

Therefore, entries k_1, k_2, and k_3 exist such that $k_1 \rightarrow k_2$ and $k_2 \Rightarrow k_3$ where k_1 is in $C(e_1)$, k_2 is in E, and k_3 is in $R(e_1)$.

Suppose $n \geq 6$ and $|\beta| \geq 1$ when $e_4 \neq j-1$. Then we showed that $p \geq n-|\beta|-4 \geq 1$. Since b_2 $(= i+1)$ is faulty, we have $q \geq 1$. Also, we have $q'' = |\beta|+q+p \leq n-3$. If $q \geq 2$, then we have $p \leq n-|\beta|-5$, contradicting the assumption that $p \geq n-|\beta|-4$. If $q = 1$, and thus $p = 1$, then $p \leq n - |\beta| - 4 \leq n - 5$ and $n = 6$. In this case, it is possible that we cannot find a non-faulty unused entry $e_8(\neq \overline{e_2})$ in $C(e_1)$ or $C(\overline{e_1})$. Also, it is possible that we cannot find non-faulty entries $k_1 \rightarrow k_2$ and $k_2 \Rightarrow k_3$ such that k_1 is in $C(e_1)$, k_2 is in E, and k_3 is in $R(e_1)$. In this case, we call BFS(M) to find five non-faulty entries such that no two entries are common, and are in the same row or column. The algorithm BFS(M) is a modification of the standard breadth-first search algorithm and finds a set E of size $\min\{deg_f(u), deg_f(v)\}$. The existence of such five entries can be examined by using the following example, where entries are as in the above discussion: first we find $\overline{e_1}$, $\overline{e_2}$, and then $\overline{e_7}$ in $R(e_3)$. It is easy to see that they are not common, and are in different rows and columns. Note that $\overline{e_7}$ must be in $R(e_3)$ because $p = 1$, and there is one used entry in $R(e_3)$ which is not $\overline{e_3}$. Next, we find an entry d in $R(e_1)$ and $C(e_2)$. The entry d is non-faulty and unused because $\overline{d}$ is in $C(\overline{e_1})$, or d and $\overline{d}$ are in the same row. Finally, we find an entry d' in an unused row and column (which is $C(e_7)$), where $d' \neq 0$ because an entry in $C(e_7)$ with a value 0 is in $R(\overline{e_1})$.

Thus, unless $e_4 \neq j-1$ and $n = 6$, there is an entry e_6 such that $e_3 \rightarrow e_6$ where e_6 is unused or $\overline{e_3}$. If $C(e_6)$ is used by e_7 in E, then we can find a non-faulty entry e_8 in $C(\overline{e_1})$ such that $e_7 \Rightarrow e_8$, and e_8 is unused, or e_8 and e_7 are common. Therefore, we can add e_1, e_2, e_6, and e_8 to E after removing $\overline{e_1}$, e_3, and e_7 from E.

Case 3.2.2. e_5 is faulty and $e_4 = 0$.

Since there are at most $(q + |\beta|) + (n - q - |\beta| - 4) = n - 4$ faulty entries in $R(\overline{e_1})$ and $C(\overline{e_1})/\alpha$, there is at most one faulty entry in $R(e_3)/\beta$ except e_5. Suppose there is one faulty entry $e_5'(\neq e_5)$ is in $R(e_3)/\beta$. Then $p \geq n - |\beta| - 5$ because we exclude entries e_3, e_4, e_5, e_5', and a used entry e' such that e' and e_3 are not common, where $1 \leq |\beta| \leq n - 3 - 5 = n - 8$. Thus, unless $n \leq 6$ and $|\beta| = 1$, we have $p \geq 1$, and we can find at least one non-faulty unused entry e_6, or a used entry e_6 such that e_6 and e_3 are common. If $n \leq 6$, $|\beta| = 1$, and $b_2 = i + 1$, then entries b_2, e_5, and e_5' are faulty. Since $|\alpha| \geq 1$ and $|\beta| \geq 1$, in M, there are at least $n - 1$ faulty nodes in Q_n. It contradicts the assumption that there are at most $n - 2$ faulty nodes in Q_n. If $n \leq 6$, $|\beta| = 1$, and $b_2 = 0$, then the number of faulty entries in $R(\overline{e_1})$ and $C(\overline{e_1})/\alpha$ is at least $(q+|\beta|)+(n-q-|\beta|-3) = n-3$. Since $|\alpha| \geq 1$, and e_5' is faulty, there are at least $n - 1$ faulty nodes in Q_n. It again leads to the contradiction. It shows $p \geq 1$. In addition, the entry $\overline{e_5}$ must be in $R(\overline{e_1})$. If $C(e_6)$ is used by e_7, then $q'' = |\beta| + q + p \leq n - 4$. If $b_2 = i + 1$, then entries $\overline{e_5}$ and b_2 are faulty. Thus, $q \geq 2$, and we have $p \leq n - |\beta| - 6$. It contradicts the assumption that $p \geq n - |\beta| - 5$. If $b_2 = 0$, then the number of faulty entries in $R(\overline{e_1})$ and $C(\overline{e_1})/\alpha$ is at least $n-3$. Since $|\alpha| \geq 1$ and e_5' is faulty, there are at least $n-1$ faulty nodes in Q_n, which again leads to a contradiction. It shows that we can find a non-faulty entry e_8 in $C(\overline{e_1})$ such that $e_7 \rightarrow e_8$, and e_8 is unused, or e_8 and e_7 are common.

Suppose there is no such faulty entry e_5'. Then $p \geq n - |\beta| - 5$ because we exclude entries e_3, e_4, e_5, $\overline{e_5}$, a used entry e' such that e' and e_3 are not common, where $1 \leq |\beta| \leq n - 3 - 5 = n - 8$. Unless $b_2 = i + 1$, $e_5 = i - 1$, $n = 6$ and $|\beta| = 1$, all discussion above can be applied similarly. If $b_2 = n + 1$, $e_5 = i - 1$, $n = 6$, and $|\beta| = 1$, then it is possible that there is only one used entry e_6 in $R(e_3)$ which is used by $\overline{e_6}$ $(\neq e_3)$. Also, it is possible that we cannot find non-faulty entries k_1, k_2, and k_3 such that $k_1 \rightarrow k_2$ and $k_2 \Rightarrow k_3$ where k_1 is in $C(e_1)$, k_2 in E, and k_3 in $R(e_1)$. In this case, we call BFS(M) to find five non-faulty entries such that no two entries are common, and are in the same row or column. The existence of such entries is shown by the following example, where entries are as in the above discussion: First, we find e_1, e_2, and e_6, and then d in $R(\overline{e_6})$ and $C(\overline{e_1})$. The entry d is non-faulty and unused because $\overline{d}$ is in $R(e_1)$, or d and $\overline{d}$ are in the same row $R(\overline{e_6})$. Finally, we find an entry d' in unused row and column (which is $C(\overline{e_6})$). For the entry d', $d' \neq 0$ because an entry in $C(\overline{e_6})$ with a value 0 is in $R(\overline{e_1})$. Moreover, d' is non-faulty and unused because $\overline{d'}$ is in $R(e_3)$ and d' and e_6 are not common.

Case 3.2.3. e_5 and e_4 are faulty.

Since there are at least $(q+|\beta|)+(n-q-|\beta|-3) = n-3$ faulty entries in $R(\overline{e_1})$ and $C(\overline{e_1})/\alpha$, there is no faulty entry in $R(e_3)/\beta$ except e_4 and e_5. Also, $|\alpha| = 1$ and $\overline{e_5}$ must be in $R(\overline{e_1})$. If $e_4 = j-1$, then $p \geq n-|\beta|-6$ because we exclude a entry with a value 0, e_3, e_4, $\overline{e_4}$, e_5, and a used entry e' such that e' and e_3 are not common, where $1 \leq |\beta| \leq n-3-6 = n-9$. If $e_4 \neq j-1$, then $p \geq n-|\beta|-5$ because we exclude a entry with a value 0, e_3, e_4, e_5, and a used entry e' such that e' and e_3 are not common, where $1 \leq |\beta| \leq n-3-5 = n-8$. Thus, unless $n \leq 7$ and $|\beta| = 1$ when $e_4 = j-1$, and unless $n \leq 6$ and $|\beta| = 1$ when $e_4 \neq j-1$, we have $p \geq 1$ and, in $R(e_3)$, there exist at least one non-faulty unused entry e_6, or a used entry e_6 such that e_6 and e_3 are common. In addition, $b_2 \neq 0$ because if not, the number of faulty entries in $R(\overline{e_1})$ and $C(\overline{e_1})/\alpha$ is at least $(q+|\beta|)+(n-q-|\beta|-2) = n-2$. Since $|\alpha| = 1$, there are at least $n-1$ faulty nodes in Q_n. If $n \leq 6$ and $|\beta| = 1$, then entries b_2, e_4, and e_5 are faulty. Since $|\alpha| = 1$ and $|\beta| = 1$, there are at least $n-1$ faulty nodes in Q_n. It contradicts the assumption that there are at most $n-2$ faulty nodes in Q_n. Thus, unless $e_4 = j-1$, $n = 7$, and $|\beta| = 1$, we can find at least one non-faulty unused entry e_6 or a non-faulty entry $e_6(= \overline{e_3})$. If $e_4 = j-1$, $n = 7$, and $|\beta| = 1$, then we can show non-faulty entries k_1, k_2, and k_3 exist such that $k_1 \rightarrow k_2$

and $k_2 \Rightarrow k_3$ where k_1 is in $C(e_1)$, k_2 is in E, and k_3 is in $R(e_1)$.

If $C(e_6)$ is used by e_7 in E, then we continuously search a non-faulty entry e_8 in $C(e_1)$ or $C(\overline{e_1})$. Suppose $n \geq 8$ and $|\beta| \geq 1$ when $e_4 = j-1$. Then we showed $p \geq n - |\beta| - 6 \geq 1$. In this case, $q'' = |\beta| + q + p \leq n-3$. If $q \geq 4$, then we have $p \leq n - |\beta| - 7$, which contradicts the assumption that $p \geq n - |\beta| - 6$. Thus, we can find at least one non-faulty entry e_8 in $C(\overline{e_1})$. If $q \leq 3$, then there exist non-faulty entries k_1, k_2, and k_3 such that $k_1 \rightarrow k_2$ and $k_2 \Rightarrow k_3$ where k_1 is in $C(e_1)$, k_2 is in E, and k_3 is in $R(e_1)$. Suppose $n \geq 7$ and $|\beta| \geq 1$ when $e_4 \neq j-1$. Then we showed $p \geq n - |\beta| - 5 \geq 1$. Again, $q'' = |\beta| + q + p \leq n-3$. If $q \geq 3$, then we have $p \leq n - |\beta| - 6$ which again leads to the contradiction. If $q = 2$, then $p = 1$ or 2. Since $p \leq n - |\beta| - 5 \leq n-7$, we have $n = 7$ or 8. If $n = 8$, then there are at least two non-faulty entries ($\neq \overline{e_1}$) in $R(\overline{e_1})$. Since $q = 2$, for a non-faulty entry e_2 in $R(\overline{e_1})$, we can find entries e_4 and e_5 such that $e_2 \Rightarrow e_3$, $e_3 \rightarrow e_4$, and $e_3 \rightarrow e_5$, where $e_5 = 0$ or e_5 is non-faulty, which does not fall into case 3.2.3. Thus, it suffices to show the case $n = 7$ when $q = 2$. Unless non-faulty entries k_1, k_2, and k_3 exist such that $k_1 \rightarrow k_2$ and $k_2 \Rightarrow k_3$ where k_1 is in $C(e_1)$, k_2 is in E, and k_3 is in $R(e_1)$, we call BFS(M) to find six non-faulty entries such that no two entries are common, and are in the same row or column. The existence of such entries is given by the following example, where entries are as in the above discussion: First, we find entries $\overline{e_1}$, e_3, and e_7, and then an entry d in $R(e_1)$ and $C(e_6')$ where e_6' is a used entry in $R(e_3)$, and e_6' and e_3 are not common. The entry d is non-faulty and unused because $\overline{d}$ is in $C(\overline{e_1})$, or d and $\overline{d}$ are in the same row $R(e_1)$. Next, we find an entry d' in $R(\overline{e_6'})$ and $C(e_1)$. The entry d' is non-faulty and unused because $\overline{d'}$ is in $R(\overline{e_1})$, or d' and $\overline{d'}$ are in the same row $R(\overline{e_6'})$. Finally, we find an entry d'' in an unused row and column (which is $C(\overline{e_6'})$). We need to show d'', e_3, and e_7 are not common. Since $R(\overline{e_2})$ is $\overline{C(e_2)}$, the entry $\overline{e_3}$ is in $R(\overline{e_2})$, where we already found the entry e_7. Thus, d'' and e_3 are not common.

Also, since $\overline{R(e_6')}$ is $C(\overline{e_6'})$, $\overline{d''}$ is in $R(e_6')$ (which is also $R(e_3)$). Thus, d'' and e_7 are not common. Therefore, unless $e_4 \neq j-2$ and $n=7$, we can find a non-faulty entry e_8 in $C(\overline{e_1})$ such that $e_7 \to e_8$, and e_8 is unused, or e_8 and e_7 are common. Now, we can add e_1, e_2, e_6, and e_8 to E after removing $\overline{e_1}$, e_3, and e_7 from E.

From the above discussion, we show that for the given set E of non-faulty entries, feasible and unused row and column with the used entry in an algorithm **Aug-II**, the algorithm **Aug-II** increases the size of E by one, or finds $\min\{deg_f(u), deg_f(v)\}$ non-faulty entries such that no two entries are common, and are in the same row or column. □

Lemma D.4 *For a given set E of entries such that no two entries are common, and are in the same row or column, feasible and unused row R and column C, and an entry e_1 in R and C, if the entry e_1 is used by $\overline{e_1}$ such that $\overline{e_1}$ is in E, then the algorithm* **Augmenting-I** *increases the size of E by one, or finds the set E of size $min\{deg_f(u), deg_f(v)\}$.*

PROOF. Let $C(e_1)$ be a column i, b_1 be an entry in $R(e_1)$ and $C(\overline{e_1})$, and b_2 be an entry in $R(\overline{e_1})$ and $C(e_1)$.

Case 1. $\overline{e_1}$ is the only non-faulty entry in $R(\overline{e_1})$.

We use the algorithm **Aug-I** to increase the size of E by one. It is verified by Lemma D.2.

Case 2. both entries b_1 and b_2 are non-faulty.

Let $e_1 = j$. Then $b_1 = j+1$ and $b_2 = i+1$. In this case, b_1 can be added to E after removing $\overline{e_1}$ from E. Also, $b_1 = j+1$ and $b_2 = i+1$, $i \neq j$, are not common, and they are not in the same row or column. Thus, we can add b_2 to E, and the size of E increases by one because we remove $\overline{e_1}$ from E, and add b_1 and b_2.

Case 3. b_1 is non-faulty, but b_2 is 0 or faulty.

Since there are at least two non-faulty entries in a column of M, we can find a non-faulty entry k_1 ($\neq e_1$) in $C(e_1)$. Suppose there are non-faulty entries k_1, k_2, and k_3 such that $k_1 \rightarrow k_2$ and $k_2 \Rightarrow k_3$ where k_1 is in $C(e_1)$, k_2 is E, and k_3 is in $R(e_1)$. If $k_1 = i - 1$, then k_1 and $\overline{k_1}$ are in the same row. Thus, if k_1 is used, then $\overline{k_1}$ and k_2 are common. If $k_1 \neq i - 1$, then $\overline{k_1}$ is in $R(\overline{e_1})$ and k_1 is unused. Thus, the entry k_1 can be added to E after removing k_2 from E. Since the entry $\overline{k_3}$ is in $C(\overline{e_1})$, k_3 is unused, and we can add k_3 to E. If there are no such non-faulty entries k_1, k_2, and k_3, then we use the algorithm **Aug-II** to increase the size of E by one, or to find $\min\{deg_f(u), deg_f(v)\}$ non-faulty entries in M such that no two entries are common, and are in the same row or column. It is verified by Lemma D.3.

Case 4. b_1 is 0 or faulty.

If b_1 is 0 or faulty but b_2 is non-faulty, then we exchange e_1 and $\overline{e_1}$, and then apply the algorithm **Augmenting-I**. That is, we set $E = E - \{\overline{e_1}\}$ and $E = E \cup \{e_1\}$. Since removing $\overline{e_1}$ from E makes $R(\overline{e_1})$ and $C(\overline{e_1})$ unused, we can apply the algorithm **Augmenting-I** with an entry $\overline{e_1}$ in $R(\overline{e_1})$ and $C(\overline{e_1})$. For other cases, we apply the algorithm **Aug-II** with a slight modification in step 3.3. That is, in step 3.3, we rule out cases 1 and 2, and then apply the algorithm used for case 3. Similar to Lemma D.3, we can show that the correctness of step 5.2 of the algorithm **Augmenting-I**.

From the above discussion, we show that the algorithm **Augmenting-I** increases the size of E by one, or finds $\min\{deg_f(u), deg_f(v)\}$ non-faulty entries in M such that no two entries are common, and are in the same row or column. □

So far, we discussed the correctness of the algorithm **Augmenting-I**. In the following lemma, we continue discussing about the correctness of the algorithm **Augmenting-**

Algorithm. Augmenting-II(E, R, C)

Input: a set E of Δ entries such that no two entries are common, and are in the same row or column, feasible and unused row R and column C.

Output: the set E of size $\Delta + 1$, or of size $\min\{deg_f(u), deg_f(v)\}$.

{Assumption: in C, all non-faulty entries $e_1, e_2, \ldots, e_h$ are in rows used by $e'_1, e'_2, \ldots, e'_h$ such that for all i, $1 \le i \le h$, e'_i is in E, and $e_i \rightarrow e'_i$, respectively. }

1. let e''_i be an entry in R such that $e'_i \Rightarrow e''_i$;
2. **if** there is a non-faulty entry e''_i in R
   then call Aug-III(E, R, C);
3. **else** call Aug-IV(E, R, C);

Fig. 10. The algorithm **Augmenting-II**

II, which is used in the algorithm **Augmenting**. The algorithm **Augmenting-II** is given in Fig. 10. Recall that we use the algorithm **Augmenting-II** to handle the case that all non-faulty entries in an unused column are in used rows. To show the correctness of the algorithm **Augmenting-II**, we discuss the correctness of the algorithms **Aug-III** given in Fig. 11 and **Aug-IV** given in Fig. 12 by going through each step.

First, we show how we can find the unused row R and column C such that R is not $\overline{C}$. For feasible and unused row R and column C, suppose $\overline{R}$ is C. Then in a row R' which is different to R, we find a non-faulty entry e_j such that $e_j \rightarrow e'_j$, e'_j is in E, and $C(e'_j)$ is not $\overline{R}$. If both e_j and $\overline{e_j}$ are not in R', then $\overline{e_j}$ is in R. Since the entry $\overline{e_j}$ in R is unused, we can add e_j after removing e'_j from E. If both e_i and $\overline{e_j}$ are in R', then regardless of whether e_j and e'_j are common or not, we can add e_j to E after removing e'_j from E. Now, $C(e'_j)$ becomes feasible and unused. Thus, we let $C(e'_j)$ be C. In this way, we have row R and column C such that $\overline{R}$ is not C. Note that once we set $C(e'_j)$ to be C, all non-faulty nodes in C may not be used, or are in used rows. That is, we cannot directly call the algorithm **Augmenting**. Thus, we execute step 3.3 again before calling the algorithm **Augmenting**(E, R, C) in step 3.4 of the

Algorithm. Aug-III(E, R, C)

Input: a set E of Δ entries such that no two entries are common, and are in the same row or column, feasible and unused row R and column C.

Output: the set E of size $\Delta + 1$, or of size $\min\{deg_f(u), deg_f(v)\}$.

{Assumption: for non-faulty entries e_i in C, e'_i in E, and e''_i in R, $1 \leq i \leq h$ such that $e_i \rightarrow e'_i$ and $e'_i \Rightarrow e''_i$, we can find a non-faulty entry e''_i in R. }

1. **if** e_i is unused, or e_i and e'_i are common
1.1 **if** e''_i is unused
  then $E = E - \{e'_i\}$; $E = E \cup \{e_i, e''_i\}$;
1.2 **else** $E = E - \{e'_i\}$; $E = E \cup \{e_i\}$;
  call Augmenting-I($E, R(e''_i), C(e''_i), e''_i$);
2. **else** (e_i is used by $\overline{e_i}$ in E, and e_i and e'_i are not common)
2.1 **if** e''_i is unused
  then $E = E - \{e'_i\}$; $E = E \cup \{e''_i\}$;
  call Augmenting-I($E, R(e_i), C(e_i), e_i$);
2.2 **else**
  find a feasible row R' such that an entry in R' and C is 0 or faulty;
  find a non-faulty entry d_j in R' such that for a non-faulty entry e_j in C, $e_j \rightarrow e'_j$ and $e'_j \Rightarrow d_j$ where e_j is unused, or e_j and e'_j are common, and e'_j is in E;
  if R' is unused
  then $E = E - \{e'_j\}$; $E = E \cup \{e_j\}$;
  else (R' is used by d_k in E)
   find an entry d'_k in R such that $d_k \Rightarrow d'_k$;
   $E = E - \{e'_j, d_k\}$; $E = E \cup \{e_j, d'_k, \}$;
  if d_j is unused
  then $E = E \cup \{d_j\}$;
  else call Augmenting-I($E, R(d_j), C(d_j), d_j$);

Fig. 11. The algorithm **Aug-III**

Algorithm. Aug-IV(E, R, C)

Input: a set E of Δ entries such that no two entries are common, and are in the same row or column, feasible and unused row R and column C (which is not $\overline{R}$).

Output: the set E of size $\Delta + 1$, or of size $\min\{deg_f(u), deg_f(v)\}$.

$\{$Assumption: for non-faulty entries e_i in C, e_i' in E, and e_i'' in R, $1 \leq i \leq h$ such that $e_i \rightarrow e_i'$ and $e_i' \Rightarrow e_i''$, we cannot find a non-faulty entry e_i''.$\}$

1. find a feasible row R' such that an entry in R' and C is 0 or faulty;
2. find a non-faulty entry d_i in R' such that for a non-faulty entry e_i in C, $e_i \rightarrow e_i'$ and $e_i' \Rightarrow d_i$ where e_i is unused, or e_i and e_i' are common, and e_i' is in E;
3. **if** R' is unused
3.1 **then** $E = E - \{e_i'\}$; $E = E \cup \{e_i\}$;
3.2 **if** d_i is unused
then $E = E \cup \{d_i\}$;
3.3 **else** call Augmenting-I($E, R(d_i), C(d_i), d_i$);
4. **else** (R' is used by d_k in E)
4.1 let d_k' be an entry in R such that $d_k \Rightarrow d_k'$,
4.2 find an entry e_j' in E such that for a non-faulty entry e_j in C, $e_j \rightarrow e_j'$ and $\overline{R}$ is $C(e_j')$;
4.3 find non-faulty entries k_1, k_2, and k_3 such that $e_j' \Rightarrow k_1$, $k_1 \rightarrow k_2$ and $k_2 \Rightarrow k_3$ where k_2 is in E, and k_3 is in R';
4.4 **if** d_k' is unused
then $E = E - \{e_i', d_k\}$; $E = E \cup \{e_i, d_k'\}$;
if d_i is unused
then $E = E \cup \{d_i\}$;
else call Augmenting-I($E, R(d_i), C(d_i), d_i$);
4.5 **else**
$E = E - \{e_j', k_2, d_k\}$; $E = E \cup \{e_j, k_1, d_k'\}$;
if k_3 is unused
then $E = E \cup \{k_3\}$;
else call Augmenting-I($E, R(k_3), C(k_3), k_3$);

Fig. 12. The algorithm **Aug-IV**

algorithm **Parallel-Edge-Paring**.

Lemma D.5 *For a given set E of entries such that no two entries are common, and are in the same row or column, feasible and unused row R and column C of M, if there is a non-faulty entry e''_i in R such that $e_i \rightarrow e'_i$ and $e'_i \Rightarrow e''_i$ where e_i is non-faulty and is in C, and e'_i is in E, then the algorithm* **Aug-III** *increases the size of E by one, or finds the set E of size $min\{deg_f(u), deg_f(v)\}$.*

PROOF. Suppose, in C, all non-faulty entries $e_1, e_2, \ldots, e_h$ are in rows used by $e'_1, e'_2, \ldots, e'_h$ such that for all e_i, $1 \leq i \leq h$, $e_i \rightarrow e'_i$ and e'_i is in E, respectively. Also, suppose there is a non-faulty entry e''_i in R such that $e'_i \Rightarrow e''_i$. Let g be the number of faulty entries in C/α, and b_1 be an entry in R and C.

We first show that there is at least one non-faulty entry e''_i in R when $b_1 = 0$, or there are only two non-faulty entries in C. Note that the entry b_1 is 0 or faulty because we assume that all non-faulty entries in C are in used rows. By way of contradiction, assume that there is no such non-faulty entry e''_i. Suppose $b_1 = 0$, then there are at least $n - g - |\alpha| - 2$ faulty entries in R/β. Thus, in Q_n, there are at least $|\alpha| + |\beta| + g + (n - g - |\alpha| - 2) = n + |\beta| - 2 \geq n - 1$ faulty nodes. It contradicts the assumption that there are at most $n - 2$ faulty nodes in Q_n. Suppose there are only two non-faulty entries in C. That is, $h = 2$. Then $|\alpha| + g \geq n - 3$, and there is at least one faulty entry in R/β. Thus, in Q_n, there are at least $n - 3 + |\beta| + 1 \geq n - 1$ faulty nodes. It shows again the contradiction. Therefore, if $b_1 = 0$ or $h = 2$, then we can find at least one non-faulty entry e''_i in R such that $e'_i \Rightarrow e''_i$. Specifically, if $h = 2$, then there are at least two non-faulty entries e''_i and e''_h, $1 \leq i, j \leq h$ in R. In this case, one of entries must be unused.

Step 1 of the algorithm

Suppose e_i is unused, or e_i and e'_i are common. If e''_i is unused, then we can add entries e_i and e''_i to E after removing e'_i from E. Thus, the size of E increases by one. If e''_i is used, then we add e_i to E after removing e'_i from E. Now, the used entry e''_i is in an unused row and column. Thus, we can call **Augmenting-I**$(E, R(e''_i), C(e''_i), e''_i)$. From Lemma D.4, the algorithm **Augmenting-I** increases the size of E by one, or finds the set E of size $\min\{deg_f(u), deg_f(v)\}$. Therefore, the lemma holds.

Step 2 of the algorithm

Suppose e_i is used, and e_i and e'_i are not common.

Step 2.1 of the algorithm

Suppose e''_i is unused. Then we can add e''_i to E after removing e'_i from E. Now, the used entry e_i is in an unused row and column, so we can call **Augmenting-I**$(E, R(e_i), C(e_i), e_i)$. From Lemma D.4, the lemma holds.

Step 2.2 of the algorithm

Suppose e''_i is used, then we find a feasible row R' such that an entry in R' and C is 0 or faulty. If there is no such feasible row, then there are $n - |\alpha|$ non-faulty entries in C. It implies we already found a set E of size $n - |\alpha| = \min\{deg_f(u), deg_f(v)\}$. Thus, we assume that we can find a feasible row R'. Also, for a non-faulty entry e_j in C, we find a non-faulty entry d_j in R' such that $e_j \rightarrow e'_j$ and $e'_j \Rightarrow d_j$ where e_j is unused, or e_j and e'_j are not common, and e'_j is in E. To show the existence of a non-faulty entry d_j, we consider h entries d_j, $1 \leq j \leq h$, in R' such that $e_j \rightarrow e'_j$ and $e'_j \Rightarrow d_j$ where $h \geq 3$, e_j is non-faulty and in C, and e'_j is in E. Then there are at least two such non-faulty entries d_{j_1} and d_{j_2}, $1 \leq j_1, j_2 \leq h$ in R' because there is at most one faulty entry in R'. Recall that if $h = 2$ then we can find an unused entry e''_j in R. Thus, there are non-faulty entries e_{j_1} and e_{j_2} in C such that $e_{j_1} \rightarrow e'_{j_1}$ and $e'_{j_1} \Rightarrow d_{j_1}$, and $e_{j_2} \rightarrow e'_{j_2}$ and $e'_{j_2} \Rightarrow d_{j_2}$ where e'_{j_1} and e'_{j_2} are in E. If both e_{j_1} and e_{j_2} are used, then either e_{j_1} is used, and e_{j_2} and e'_{j_2} are common, or e_{j_2} is used, and e_{j_1}

and e'_{j_1} are common. Otherwise, one of entries e_{j_1} and e_{j_2} is unused. It shows that we can find a non-faulty entry d_j in R' such that $e_j \rightarrow e'_j$ and $e'_j \Rightarrow d_j$ where e_j is in C and e'_j is in E.

Now, if R' is unused, then we can add e_j to E after removing e'_j from E. We again add d_j to E if d_j is unused, or call **Augmenting-I**$(E, R(d_j), C(d_j), d_j)$ if d_j is used. If R' is used by an entry d_k ($\neq d_i$, $1 \leq i \leq h$) then there is a non-faulty entry d'_k in R such that $d_k \Rightarrow d'_k$. Since we assume that e''_i in R is used, the entry d'_k in R cannot be used. Thus, we can add entries e_j and d'_k to E after removing entries e'_j and d_k from E. Similarly, we add d_j to E if d_j is unused, or call **Augmenting-I**$(E, R(d_j), C(d_j), d_j)$ if d_j is used. Thus, the lemma holds.

From the above discussion, we show that the algorithm **Aug-III** increases the size of E by one, or finds the set E of size $\min\{deg_f(u), deg_f(v)\}$. □

Lemma D.6 *For a given set E of entries such that no two entries are common, and are in the same row or column, feasible and unused row R and column C of M, if, there is no non-faulty entry e''_i in R such that $e_i \rightarrow e'_i$ and $e'_i \Rightarrow e''_i$ where e_i is non-faulty and is in C, and e'_i is in E, then the algorithm* **Aug-IV** *increases the size of E by one, or finds the set E of size $min\{deg_f(u), deg_f(v)\}$.*

Proof. In Lemma D.5, we showed that there is at least one-non-faulty entry e''_i in R if $b_1 = 0$ or there are only two non-faulty entries in C, where b_1 is an entry in R and C. Thus, we assume that b_1 is faulty, and there are at least three non-faulty entries in C. Let g be the number of faulty entries in C/α.

Step 1 of the algorithm

Suppose, in C, all non-faulty entries $e_1, e_2, \ldots, e_h$ are in rows used by $e'_1, e'_2, \ldots, e'_h$ such that for all e_i, $1 \leq i \leq h$, $e_i \rightarrow e'_i$ and e'_i is in E, respectively. Also, suppose we cannot find a non-faulty entry e''_i in R such that $e'_i \Rightarrow e''_i$. Then we find a feasible row R' such that an entry in R' and C is 0 or faulty.

Step 2 of the algorithm

Similar to step 2.2 of the algorithm **Aug-III**, we can find a non-faulty entry d_i in R' such that for a non-faulty entry e_i in C, $e_i \rightarrow e'_i$ and $e'_i \Rightarrow d_i$ where e_i is unused, or e_i and e'_i are common, and e'_i is in E.

Step 3 of the algorithm

Suppose R' is unused. Then we can add e_i to E after removing e'_i from E. We again add d_i to E if d_i is unused, or call **Augmenting-I** $(E, R(d_i), C(d_i), d_i)$. Thus, the lemma holds.

Step 4 of the algorithm

Suppose R' is used by d_k in E, then we can find an entry d'_k in R such that $d_k \Rightarrow d'_k$. First, we find an entry e'_j in E such that $e_j \rightarrow e'_j$, e'_j is in C, and $\overline{R}$ is $C(e'_j)$. We assume that we cannot find a non-faulty entry e''_i in R such that $e_i \rightarrow e'_i$ and $e'_i \Rightarrow e''_i$ where e_i is non-faulty and is in C, and e'_i is in E. Thus, there are at least $n - g - |\alpha| - 2$ faulty entries R, where we assume that among entries e''_i, $1 \leq i \leq h$, an entry has a value 0, and two entries, say e''_j and e''_k, are common. Thus, in Q_n, there are at least $|\alpha| + |\beta| + g + (n - g - |\alpha| - 3) = n - 2$ faulty nodes. In this case, $\overline{R}$ is $C(e'_j)$ or $C(e'_k)$ because if two entries e and e' in the same row are common, then $\overline{R(e)}$ is $C(e)$ or $C(e')$. Without loss of generality, assume that $\overline{R}$ is $C(e'_j)$.

Next, we find non-faulty entries k_1, k_2, and k_3 such that $e'_j \Rightarrow k_1$, $k_1 \rightarrow k_2$, and $k_2 \Rightarrow k_3$ where k_2 is in E, and k_3 is in R'. To show the existence of such non-faulty entries k_1, k_2, and k_3, we consider $h - 1$ entries d_g $(\neq e'_j)$, $1 \leq g \leq h - 1$, in $C(e'_j)$ such that an entry in $R(d_g)$ and C is non-faulty. Since there are $h - 3$ faulty entries in $C(e'_j)$, there exists at least one non-faulty entry d_g such that $e'_j \Rightarrow d_g$. Since a non-faulty entry in C is in a used row, the entry d_g is in a row used by an entry, say

d'_g. What remains is to show that we can find a non-faulty entry d''_g in R' such that $d'_g \Rightarrow d''_g$. If $\overline{R'}$ is not C, then there is no faulty entry in R'/β. Also, $d_j = 0$ because we assume d_j is not non-faulty. Thus, d''_g is non-faulty. If $\overline{R'}$ is C, then the number of non-faulty entries in C must be $n - |\alpha| - 2$. Otherwise, we can find a row R' such that $\overline{R'}$ is not C. Also, an entry b'_1 in R' and C must be 0, and entries in R'/β except d_j are non-faulty. Thus, d''_g is non-faulty. It shows that we can find non-faulty entries k_1, k_2, and k_3 such that $e'_j \Rightarrow k_1$, $k_1 \rightarrow k_2$, and $k_2 \Rightarrow k_3$ where k_2 is in E, and k_3 is in R'.

If d'_k is unused, then we can add e_i and d'_k to E after removing e'_i and d_k from E. We again add d_i to E if d_i is unused, or call **Augmenting-I** $(E, R(d_i), C(d_i), d_i)$. If d'_k is used, then entries e'_j and d'_k must be common. Also, k_1 is unused because $\overline{k_1}$ is in R which is unused. Thus, we can add e_j, k_1, and d'_k to E after removing e'_j, k_2, and d_k from E. We again add k_3 to E if k_3 is unused, or call **Augmenting-I** $(E, R(k_3), C(k_3), k_3)$. Thus, the lemma holds.

From the above discussion, we show that the algorithm **Aug-IV** increases the size of E by one, or finds the set E of size $\min\{deg_f(u), deg_f(v)\}$. □

Lemma D.7 *For a given set E of entries such that no two entries are common, and are in the same row or column, feasible and unused row R and column C of M, if all non-faulty entries in C are in used rows, then the algorithm* **Augmenting-II** *increases the size of E by one, or finds the set E of size $min\{deg_f(u), deg_f(v)\}$.*

The following theorem directly comes from Lemma D.4 and Lemma D.7.

Theorem D.8 *For a given set E of entries such that no two entries are common, and are in the same row or column, the algorithm* **Augmenting** *increases the size of E by one, or finds the set E of size $min\{deg_f(u), deg_f(v)\}$.*

Algorithm. Parallel-Routing-Cube

Input: non-faulty nodes $u = 1^r 0^{n-r}$ and $v = 0^n$ in Q_n with at most $n-2$ faulty nodes.

Output: $\min\{deg_f(u), deg_f(v)\}$ parallel fault-free paths of length $\leq dist(u,v)+4$ from u to v.

1. **case 1.** u and v have no faulty neighbors
   use **Parallel-Routing-Cube-I**
2. **case 2.** u or v has faulty neighbors
   case 2.1. only one of u or v has faulty neighbors
   use **Parallel-Routing-Cube-I**, regarding u and v have no faulty neighbors;
   discard paths including faulty neighbors of u or v;
   case 2.2. both u and v have faulty neighbors
   for each entry e_k given by **Parallel-Edge-Pairing do**
   find a corresponding paired edge $([u_i, u_{i,i'}], [v_{j,j'}, v_j])$
   given by **Prematch-III**;
   construct a path of the form $u_i\langle i', \ldots j'\rangle v_j$
   by **Procedure-III**;

Fig. 13. Parallel routing on the hypercube network with faulty nodes

E. Parallel Routing Algorithm on Faulty Hypercube Networks

First, consider the lower bound of the length of the $\min\{deg_f(u), deg_f(v)\}$ node-disjoint paths from a node $u = 1^r 0^{n-r}$ to a node $v = 0^n$ in hypercube Q_n. Suppose a neighbor node of u, $u_i, r+1 \leq i \leq n$ be non-faulty, and we want to find a path from u to v via u_i. Assume that all neighbors of u_i are faulty except two nodes u and $u_{i,i'}, r+1 \leq i'(\neq i) \leq n$. Then a fault-free path of the form $u\langle i, i', \ldots\rangle v$ from u to v has length at least $dist(u,v)+4$. Thus, the length of the $\min\{deg_f(u), deg_f(v)\}$ disjoint paths from u to v is at least $dist(u,v)+4$.

For two non-faulty nodes $u = 1^r 0^{n-r}$ and $v = 0^n$ in Q_n, our algorithm constructs $\min\{deg_f(u),\ deg_f(v)\}$ node-disjoint fault-free paths from u to v such that the length of the paths is bounded by $dist(u,v)+4$. The algorithm called **Parallel-Routing-cube** is given in Fig. 13.

We summarize all these discussions in the following theorem.

Theorem E.1 *If the hypercube network Q_n has at most $n-2$ faulty nodes, then for each pair of non-faulty nodes u and v in Q_n, in time $O(n^2)$ the algorithm* **Parallel-Routing-Cube** *constructs* $\min\{deg_f(u), deg_f(v)\}$ *node-disjoint fault-free paths of length bounded by $dist(u,v)+4$ from u to v.*

PROOF. First, we discuss the length of $\min\{deg_f(u), deg_f(v)\}$ node-disjoint fault-free paths.

If there are no faulty neighbors of u and v, $\min\{deg_f(u), deg_f(v)\}$ node-disjoint paths are constructed by **Procedure-I** or **Procedure-II**. Suppose a pair (u_i, v_j) is given by **Prematch-I**. If $1 \le i \le r$, the sequence of a path P_i from u_i to v_{i-1} is a permutation of $\langle i+1, i+2, \ldots r, 1, \ldots, i-2\rangle$, and the length of the path P is $dist(u,v)$. If the sequence of a path P_i is of the form $\langle h, i+1, \ldots r, 1, \ldots i-2, h\rangle$, the length of P_i is $dist(u,v)+2$. If $r+1 \le i \le n$, the sequence of a path P_i from u_i to v_i is a permutation of $(1, \ldots r)$, and the length of P_i is $dist(u,v)+2$. If the sequence of P_i is the form $\langle h, 1, \ldots r, h\rangle$, the length of P_i is $dist(u,v)+4$. Thus, paths constructed by **Procedure-I** is a length of at most $dist(u,v)+4$.

If paths are constructed by **Procedure-II** or **Procedure-III**, the length is still at most $dist(u,v)+4$ because all paths constructed by **Procedure-II** or **Procedure-III** are constructed based on **Procedure-I**, only flipping the first or last two indices in the paths.

We now discuss the time complexity of the algorithm **Parallel-Routing-Cube**.

For each pair given by **Prematch-I**, a path is constructed by the algorithm by searching a proper path in a set of paths between them, which takes time $O(k_i * n + n)$, where k_i is the number of faulty nodes in the set of paths for the pair (u_i, v_j). If we find a fault-free and unused path of the form $u_{i'}\langle i, \ldots\rangle v_{j'}$ for a pair $(u_{i'}, v_{j'})$, $i' < i$, then mark a node $u_{i,i'}$ as a used node. In such a way, we can detect used paths in time $O(i)$ since at most $i-1$ used paths for (u_i, v_j). If all fault-free paths for the pair (u_i, v_j) include used nodes, we pick any fault-free path P for (u_i, v_j), and for the pair $(u_{i'}, v_{j'})$ that used a node P, find a new path. As we have discussed previously in detail, this happens once during the whole execution. Thus, the time complexity is bounded by $(k_1 n + \ldots + k_n n + n^2) = O(n^2)$ since the number $k_1 + \ldots k_n$ is bounded by $n-2$.

If for a pair (u_i, v_j) given by **Prematch-I**, all possible paths are blocked by faulty nodes, we simply ignore all paths constructed for other pairs $(u_{i'}, v_{j'})$, $i' < i$, and apply **Procedure-II**. Thus, it takes additional $O(n^2)$ time to construct paths for pairs given by **Prematch-II**.

For pairs given by **Parallel-Edge-Pairing**, paths are constructed by **Procedure-III**. Each step of **Augmenting-I** and **Augmenting-II** takes $O(n)$ time. Thus, in time $O(n^2)$, **Parallel-Edge-Pairing** finds non-faulty disjoint paired edges. Thus, we conclude that the time complexity for constructing paths between non-faulty neighbors of u and v is bounded by $O(n^2)$. $\square$

F. Chapter Summary

In this chapter, we have studied the strong fault tolerance of the popular hypercube networks and shown that hypercube networks are strongly fault tolerant. We have presented an algorithm of running time $O(n^2)$ that for two given non-faulty nodes u and v in a n-dimensional hypercube Q_n with at most $n-2$ faulty nodes, constructs $min\{deg_f(u), deg_f(v)\}$ node-disjoint fault-free paths from u to v such that the length of the paths is bounded by $dist(u, v) + 4$. The time complexity of our algorithm is optimal. The length of the paths constructed by our algorithm is also optimal because we can construct pairs of nodes u and v in the hypercube Q_n with $n-2$ faulty nodes for which any set of n parallel paths connecting u and v has at least one path of length $dist(u,v)+4$.

CHAPTER IV

ROUTING IN HYPERCUBE NETWORKS WITH FAULTS

A. Chapter Overview

In this chapter, we study fault tolerant routing in hypercube networks under a probability model. We develop techniques that enable us to perform formal analysis on the success probability of the routing schemes. We assume that failure probability of each node in the hypercube network is independent and every node has the same failure probability. We partition the hypercube networks into subcubes with small size and traverse subcubes to find a path between two given nodes: If the source node and destination node are in the same subcube, we use breadth-first Search to find a path in the subcube. If the source node and destination node are in different subcubes, we traverse subcubes to find a path connecting them. If we regard each subcube as a single node, then these subcubes can be found by performing dimension-order routing along subcubes. Based on this simple algorithm, we develop a routing algorithm that for two given nodes in the hypercube network, can find a fault-free path with very high probability.

B. L_2-Routing

Suppose each node in an n-cube Q_n is labeled by a distinct binary string $b_1 b_2 \cdots b_n$. Then each binary string $b_1 b_2 \cdots b_{n-k}$ of length $n-k$ corresponds to a *k-dimensional subcube* (or shortly a *k-subcube*) Q_k of 2^k nodes, where each node in Q_k is labeled as $b_1 \cdots b_{n-k} x_1 \cdots x_k$, $x_j \in \{0, 1\}$. Two k-subcubes $b_1 \cdots b_{n-k}*$ and $b'_1 \cdots b'_{n-k}*$ are *neighboring* if the strings $b_1 \cdots b_{n-k}$ and $b'_1 \cdots b'_{n-k}$ have exactly one different bit.

First, we give a brief description on our routing algorithm called L_2-**Routing**,

Algorithm. L_2**-Routing**

Input: two non-faulty nodes $u = x_1x_2 \cdots x_n$ and $v = y_1y_2 \cdots y_n$ in the n-cube network Q_n.

Output: a fault-free path in Q_n from u to v.

1. $w = u$, and initialize the path $P = [w]$;
2. **for** $i = 1$ **to** $n - k$, such that $w_i \neq y_i$ **do** $\{$assuming $w = w_1w_2 \cdots w_n\}$

2.1 **if** $w' = w_1 \cdots w_{i-1}\overline{w_i}w_{i+1} \cdots w_n$ is non-faulty
 then extend the path P to w'; let $w = w'$;

2.2 **else if** there is a j, (j is examined in the strict order $j = n - k + 1, n - k + 2, \ldots, n$) such that both $q = w_1 \cdots w_{i-1}w_iw_{i+1} \cdots w_{j-1}\overline{w_j}w_{j+1} \cdots w_n$ and $q' = w_1 \cdots w_{i-1}\overline{w_i}w_{i+1} \cdots w_{j-1}\overline{w_j}w_{j+1} \cdots w_n$ are non-faulty
 then extend the path P to q then to q'; let $w = q'$;
 else stop ('routing fails');

3. apply Breadth-first search in the k-subcube $w_1 \cdots w_{n-k} * *$ to route from w to v.

Fig. 14. The algorithm L_2-Routing

which is given in Fig. 14. L_2 is so named because for a path constructed by our algorithm from the source node to the destination node, the length of the subpath of the path connecting one node in a subcube and the other node in its neighboring subcube is bounded by 2. We assume that n-cube Q_n is decomposed into k-subcubes, where k is a positive integer less than n and can be chosen arbitrarily. For two given non-faulty nodes $u = x_1 \cdots x_n$ and $v = y_1 \cdots y_n$, suppose the Hamming distance between the substrings $x_1 \cdots x_{n-k}$ and $y_1 \cdots y_{n-k}$ is h. Then step 2 of the algorithm L_2**-Routing** traverses through $h+1$ k-subcubes Q_k^0, $\cdots$, Q_k^h, where k-subcube $Q_k^i = y_1..y_{m(i)}x_{m(i)+1} \cdots x_{n-k} * *$, $0 \leq i \leq h$ and $m(i)$ is the index of the ith different bit between u and v. The source node u is in $Q_k^0 = x_1x_2 \cdots x_{n-k} * *$, the destination node v is in $Q_k^h = y_1y_2 \cdots y_{n-k} * *$, and Q_k^{i-1} and Q_k^i are neighboring k-subcubes.

If $h = 0$, then u and v are in the same k-subcube Q_k^0. In this case, we use Breadth-first search inside Q_k^0 to find a path between u and v whose length is at most $k + 2$.

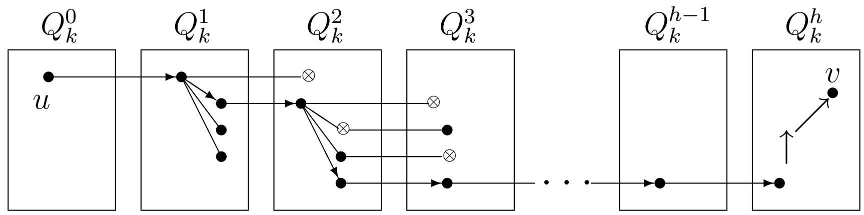

Fig. 15. Illustration of the algorithm L_2-Routing ("$\otimes$": faulty nodes, "$\bullet$": non-faulty nodes)

If $h > 0$, then we traverse k-subcubes $Q_k^0, \cdots, Q_k^h$. Suppose we arrive at a node w in Q_k^i, and the neighbor w' of w in Q_k^{i+1} is non-faulty. Then the subpath from u to v is extended by connecting the edge between w and w'. If the neighbor of w is faulty, then we find a non-faulty neighbor q of w in Q_k^i such that the neighbor q' of q in Q_k^{i+1} is also non-faulty. In this case, the subpath from u to v is extended between w and q' via q. Once we arrive at a node w in the k-subcube Q_k^h, we route from w to v within Q_k^h by using Breadth-first search.

An illustration of the algorithm L_2-**Routing** is presented in Fig. 15.

Lemma B.1 *Suppose that the algorithm L_2-***Routing** *routes successfully from u to v, then in time $O(hk + k2^k)$, L_2-***Routing** *finds a fault-free path P of length bounded by $2h + k + 2$.*

Proof.

For two given nodes $u = x_1x_2 \cdots x_n$ and $v = y_1y_2 \cdots y_n$, suppose the Hamming distance between the substrings $x_1 \cdots x_{n-k}$ and $y_1 \cdots y_{n-k}$ is h. First, we show that the length of the path P is bounded by $2h + k + 2$. The algorithm L_2-**Routing** traverses through $h + 1$ k-subcubes, $Q_k^0, \ldots, Q_k^h$, where u is in Q_k^0 and v is in Q_k^h. For two neighboring k-subcubes Q_k^{i-1} and Q_k^i, $1 \leq i \leq h$, once we arrive a node in Q_k^i, we move to the other node in Q_k^{i+1} within at most 2 hops. Thus, there are at most $2h$ hops on the path from the node u to a node w in the k-subcube Q_k^h. From the node w in Q_k^h, we route from w to v within Q_k^h by using Breadth-first Search, which takes at most $k+2$ hops. Therefore, if the algorithm L_2-**Routing** finds a fault-free path P from the node u to the node v, then the length of the path P is bounded by $2h+k+2$.

Step 2 of the algorithm L_2-**Routing** will execute at most h times. During each execution of the loop, we route from a k-subcube to its neighboring k-subcube by trying at most $k+1$ pairs of adjacent nodes. Since only at most $2(k+1)$ nodes will be tested for two neighboring subcubes, the time for the loop is bounded by $O(h(k+1)) = O(hk)$. In Step 3, Breadth-first Search takes time $O(k2^k)$. Thus, the running time of the algorithm L_2-**Routing** is bounded by $O(hk + k2^k)$. $\square$

We compute the success probability of the algorithm L_2-**Routing**. Note that each node in the n-cube Q_n belongs to a unique k-subcube in Q_n. We define an event as follows:

Event Hit(w)

The node w is contained in a k-subcube Q_k^i, and w is the first node in Q_k^i on the routing path constructed by the algorithm L_2-**Routing**.

We can extend the definition of the event **Hit** to a sequence of nodes $w_0, w_1, \ldots, w_j$ in Q_n, where $j \le h$, $w_0 = u$, and w_i is a node in the k-subcube Q_k^i, $0 \le i \le j$:

$$\mathbf{Hit}(w_0 w_1 \cdots w_j) = \bigcap_{i=1}^{j} \mathbf{Hit}(w_i)$$

That is, **Hit**$(w_0 w_1 \cdots w_j)$ is the event that w_i is the first node in Q_k^i on the routing path constructed by the algorithm L_2-**Routing** for all $1 \le i \le j$. It is easy to see that the event **Hit**$(w_0 w_1 \cdots w_j)$ uniquely determines a partial routing path constructed by

L_2-**Routing** from u to w_j. Moreover, since step 2.2 of the algorithm examines the index j in the strict order $j = n-k+1, n-k+2, \ldots, n$, for two different sequences $u, w_1, \ldots, w_j$ and $u, w'_1, \ldots, w'_j$, the events $\mathbf{Hit}(uw_1 \cdots w_j)$ and $\mathbf{Hit}(uw'_1 \cdots w'_j)$ are disjoint.

Finally, we define an event that the routing path constructed by the algorithm L_2-**Routing** can successfully reach the k-subcube Q_k^j:

$$\mathbf{Reach}(Q_k^j) = \bigcup_{w_j \in Q_k^j} \mathbf{Hit}(w_j)$$

Lemma B.2 *$Pr[\mathbf{Reach}(Q_k^j)] \geq (1 - p^{k+1}(2-p)^k)(1 - p^k(2-p)^{k-1})^{j-1}$, for all $1 \leq j \leq h$.*

Proof. A routing path constructed by the algorithm L_2-**Routing** from the node u to a node w_j in Q_k^j must go through the k-subcubes Q_k^0, Q_k^1, $\ldots$, Q_k^j. Thus, we must have:

$$\mathbf{Reach}(Q_k^j) = \bigcup_{w_1 \in Q_k^1} \cdots \bigcup_{w_j \in Q_k^j} \mathbf{Hit}(uw_1 \cdots w_j)$$

According to the definition of $\mathbf{Reach}(Q_k^j)$ and because for two different sequences $u, w_1, \ldots, w_j$ and $u, w'_1, \ldots, w'_j$, the events $\mathbf{Hit}(uw_1 \cdots w_j)$ and $\mathbf{Hit}(uw'_1 \cdots w'_j)$ are disjoint, we have:

$$Pr[\mathbf{Reach}(Q_k^j)] = \sum_{w_1 \in Q_k^1} \cdots \sum_{w_j \in Q_k^j} Pr[\mathbf{Hit}(uw_1 \cdots w_j)]$$

We prove the lemma by induction on j. Let $w_0 = u$, we have:

$$\begin{aligned} & Pr[\mathbf{Reach}(Q_k^j)] && (4.1) \\ = & \sum_{w_1 \in Q_k^1} \cdots \sum_{w_j \in Q_k^j} Pr[\mathbf{Hit}(w_0 w_1 \cdots w_j)] \end{aligned}$$

$$= \sum_{w_1 \in Q_k^1} \cdots \sum_{w_j \in Q_k^j} Pr[\mathbf{Hit}(w_j) \mid \mathbf{Hit}(w_0 w_1 \cdots w_{j-1})] \cdot Pr[\mathbf{Hit}(w_0 w_1 \cdots w_{j-1})]$$

$$= \sum_{w_1 \in Q_k^1} \cdots \sum_{w_{j-1} \in Q_k^{j-1}} Pr[\mathbf{Hit}(w_0 w_1 \cdots w_{j-1})] \cdot \sum_{w_j \in Q_k^j} Pr[\mathbf{Hit}(w_j) \mid \mathbf{Hit}(w_0 w_1 \cdots w_{j-1})]$$

We consider the probability $Pr[\mathbf{Hit}(w_j) \mid \mathbf{Hit}(w_0 w_1 \cdots w_{j-1})]$ for different nodes w_j in the k-subcube Q_k^j. We divide the discussion into two cases.

Case 1. $j = 1$, or $j \geq 2$, and w_{j-1} is adjacent to w_{j-2}.

If $j = 1$, then $w_{j-1} = u$, thus no assumption is made on the status of the neighbors of w_{j-1} in Q_k^{j-1} under the event $\mathbf{Hit}(w_0 w_1 \cdots w_{j-1}) = \mathbf{Hit}(u)$. Now suppose $j \geq 2$ and w_{j-1} is adjacent to w_{j-2}. Then in the k-subcube Q_k^{j-2}, we moved directly from w_{j-2} to w_{j-1}. Thus, step 2.2 of the algorithm L_2-**Routing** was not executed. In consequence, again, no assumption on the status of the neighbors of the node w_{j-1} in the k-subcube Q_k^{j-1} is made. In summary, in both cases, the status of the neighbors of w_{j-1} in Q_k^{j-1} is independent of the event $\mathbf{Hit}(w_0 w_1 \cdots w_{j-1})$.

Case 1.1. The distance between w_{j-1} and w_j is larger than 2; therefore, the algorithm has no way to reach the node w_j in Q_k^j under the condition $\mathbf{Hit}(w_0 w_1 \cdots w_{j-1})$. Thus, in this case, we have:

$$Pr[\mathbf{Hit}(w_j) \mid \mathbf{Hit}(w_0 w_1 \cdots w_{j-1})] = 0$$

Case 1.2. The distance from w_{j-1} to w_j is 1, i.e., w_j is adjacent to w_{j-1}. Under the assumed condition, $\mathbf{Hit}(w_j)$ if and only if w_j is non-faulty. Thus in this case we have:

$$Pr[\mathbf{Hit}(w_j) \mid \mathbf{Hit}(w_0 w_1 \cdots w_{j-1})] = 1 - p$$

Case 1.3. The distance from w_{j-1} to w_j is 2. Then there is a neighbor w'_{j-1} of w_{j-1} in Q_k^{j-1} such that w'_{j-1} and w_j are adjacent. Suppose that w'_{j-1} is the ith neighbor of w_{j-1} in Q_k^{j-1} (that is, w'_{j-1} and w_{j-1} differ by the $(n-k+i)$th bit). By the algorithm, we must have:

- the neighbor of w_{j-1} in Q_k^j is faulty;
- for each pair $\{w''_{j-1}, w''_j\}$ of nodes, where w''_{j-1} is the gth neighbor of w_{j-1} in Q_k^{j-1} and w''_j is in Q_k^j and adjacent to w''_{j-1}, $g = 1, 2, \ldots, i-1$, at least one node is faulty; and
- the ith neighbor w'_{j-1} of w_{j-1} in Q_k^{j-1} and the neighbor w_j of w'_{j-1} in Q_k^j are both non-faulty.

Therefore, in this subcase, we have the probability:

$$Pr[\mathbf{Hit}(w_j) \mid \mathbf{Hit}(w_0 w_1 \cdots w_{j-1})] = p(1-(1-p)^2)^{i-1}(1-p)^2 = p^i(1-p)^2(2-p)^{i-1}$$

Summarizing all these situations, we get:

$$\begin{aligned}
& \sum_{w_j \in Q_k^j} Pr[\mathbf{Hit}(w_j) \mid \mathbf{Hit}(w_0 w_1 \cdots w_{j-1})] \\
= \quad & Pr[\text{case 1.2}] + \sum_{i=1}^{k} Pr[\text{case 1.3, and } w'_{j-1} \text{ is the } i\text{th neighbor of } w_{j-1}] \\
= \quad & (1-p) + \sum_{i=1}^{k} p^i(1-p)^2(2-p)^{i-1} \\
= \quad & 1 - p^{k+1}(2-p)^k
\end{aligned}$$

In particular, if $j = 1$, then:

$$Pr[\mathbf{Reach}(Q_k^1)] = \sum_{w_1 \in Q_k^1} Pr[\mathbf{Hit}(w_1)] = \sum_{w_1 \in Q_k^1} Pr[\mathbf{Hit}(w_1) \mid \mathbf{Hit}(w_0)] = 1 - p^{k+1}(2-p)^k$$

Thus, the lemma is verified for $j = 1$.

Case 2. $j \geq 2$, and w_{j-1} is not adjacent to w_{j-2}.

In this case, the node w'_{j-1} in Q_k^{j-1} that is adjacent to w_{j-2} must be faulty, and the node w'_{j-2} in Q_k^{j-2} that is adjacent to w_{j-1} is non-faulty and is a neighbor

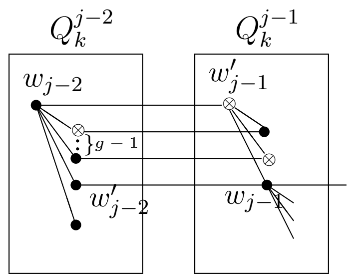

Fig. 16. The case w_{j-2} is not adjacent to w_{j-1}

of w_{j-2} (see Fig. 16 for references). Suppose that w'_{j-2} is the qth neighbor of w_{j-2} in Q_k^{j-2}. Note that the $q-1$ pairs $\{w''_{j-2}, w''_{j-1}\}$ have been checked, where w''_{j-2} is the ith neighbor of w_{j-2} in Q_k^{j-2} and w''_{j-1} is the ith neighbor of w'_{j-1} in Q_k^{j-1}, for $i = 1, \ldots, q-1$. Because a hypercube contains no cycles of length 3 [52] and w'_{j-1} and w_{j-1} are adjacent, no neighbors of w_{j-1} is a neighbor of w'_{j-1}. Therefore, besides the node w'_{j-1}, the status of the other $k-1$ neighbors of w_{j-1} in Q_k^{j-1} is independent of the event $\mathbf{Hit}(w_0 w_1 \cdots w_{j-1})$. (Note that w'_{j-1} is the qth neighbor of w_{j-1} in Q_k^{j-1}.)

Case 2.1. The distance between w_{j-1} and w_j is larger than 2. Then as before, again we have:

$$Pr[\mathbf{Hit}(w_j) \mid \mathbf{Hit}(w_0 w_1 \cdots w_{j-1})] = 0$$

Case 2.2. The distance from w_{j-1} to w_j is 1. As in Case 1.2, in this case $\mathbf{Hit}(w_j)$ if and only if w_j is non-faulty. Thus we have:

$$Pr[\mathbf{Hit}(w_j) \mid \mathbf{Hit}(w_0 w_1 \cdots w_{j-1})] = 1 - p$$

Case 2.3. The distance from w_{j-1} to w_j is 2. Then there is a neighbor x_{j-1} of w_{j-1} in Q_k^{j-1} such that x_{j-1} and w_j are adjacent. Suppose that x_{j-1} is the ith neighbor of w_{j-1} in Q_k^{j-1}. Note that $i \neq q$ since the qth neighbor of w_{j-1} in Q_k^{j-1} is w'_{j-1} which is faulty. Thus, we have:

- the neighbor of w_{j-1} in Q_k^j is faulty;
- for each pair $\{w''_{j-1}, w''_j\}$ of nodes, where w''_{j-1} is the gth neighbor of w_{j-1} in Q_k^{j-1} and w''_j is in Q_k^j and adjacent to w''_{j-1}, $g = 1, 2, \ldots, i-1$, at least one node is faulty; and
- the ith neighbor w'_{j-1} of w_{j-1} in Q_k^{j-1} and the neighbor w_j of w'_{j-1} in Q_k^j are both non-faulty.

Thus, in case $i < q$, we have:

$$Pr[\mathbf{Hit}(w_j) \mid \mathbf{Hit}(w_0 w_1 \cdots w_{j-1})] = p(1-(1-p)^2)^{i-1}(1-p)^2 = p^i(1-p)^2(2-p)^{i-1}$$

while in case $i > q$, since the qth neighbor w'_{j-1} of w_{j-1} in Q_k^{j-1} is already faulty under the condition $\mathbf{Hit}(w_0 w_1 \cdots w_{j-1})$, we have:

$$Pr[\mathbf{Hit}(w_j) \mid \mathbf{Hit}(w_0 w_1 \cdots w_{j-1})] = p(1-(1-p)^2)^{i-2}(1-p)^2 = p^{i-1}(1-p)^2(2-p)^{i-2}$$

Summarizing the above discussion, we have for Case 2 the probability:

$$\begin{aligned}
& \sum_{w_j \in Q_k^j} Pr[\mathbf{Hit}(w_j) \mid \mathbf{Hit}(w_0 w_1 \cdots w_{j-1})] \\
= \;& Pr[\text{case 2.2}] + \sum_{i=1}^{q-1} Pr[\text{case 2.3, and } w'_{j-1} \text{ is the } i\text{th neighbor of } w_{j-1}] \\
& \qquad + \sum_{i=q+1}^{k} Pr[\text{case 2.3, and } w'_{j-1} \text{ is the } i\text{th neighbor of } w_{j-1}] \\
= \;& (1-p) + \sum_{i=1}^{g-1} p^i(1-p)^2(2-p)^{i-1} + \sum_{i=g+1}^{k} p^{i-1}(1-p)^2(2-p)^{i-2} \\
= \;& 1 - p^k(2-p)^{k-1}
\end{aligned}$$

Combining the discussion in Cases 1-2, and since $1-p^{k+1}(2-p)^k \geq 1-p^k(2-p)^{k-1}$, we get:

$$\sum_{w_j \in Q_k^j} Pr[\mathbf{Hit}(w_j) \mid \mathbf{Hit}(w_0 w_1 \cdots w_{j-1})] \geq 1 - p^k(2-p)^{k-1}$$

Therefore, from the Equation (4.1), we get:

$$\begin{aligned} & Pr[\mathbf{Reach}(Q_k^j)] \\ = \; & \sum_{w_1 \in Q_k^1} \cdots \sum_{w_{j-1} \in Q_k^{j-1}} Pr[\mathbf{Hit}(w_0 w_1 \cdots w_{j-1})] \cdot \sum_{w_j \in Q_k^j} Pr[\mathbf{Hit}(w_j) \mid \mathbf{Hit}(w_0 w_1 \cdots w_{j-1})] \\ \geq \; & (1 - p^k(2-p)^{k-1}) \sum_{w_1 \in Q_k^1} \cdots \sum_{w_{j-1} \in Q_k^{j-1}} Pr[\mathbf{Hit}(w_0 w_1 \cdots w_{j-1})] \\ = \; & (1 - p^k(2-p)^{k-1}) Pr[\mathbf{Reach}(Q_k^{j-1})] \end{aligned}$$

Now by induction on j, and noticing that $Pr[\mathbf{Reach}(Q_k^1)] = 1 - p^{k+1}(2-p)^k$, we complete the proof of the lemma. □

We define the Event **Con** as follows:

Event Con(Q_k)

Each non-faulty node in a k-subcube Q_k with at most $2k-3$ faulty nodes has at least one non-faulty neighbor.

Lemma B.3 $Pr[\mathbf{Con}(Q_k)] \geq$

$$\sum_{i=0}^{k-1} p^i (1-p)^{2^k - i} \begin{pmatrix} 2^k \\ i \end{pmatrix} + \sum_{i=k}^{2k-3} p^i (1-p)^{2^k - i} \left[\begin{pmatrix} 2^k \\ i \end{pmatrix} - 2^k \begin{pmatrix} 2^k - (k+1) \\ i-k \end{pmatrix} \right]$$

Proof. Since a k-subcube Q_k has 2^k nodes, the probability that Q_k has exactly i faulty nodes is:

$$p^i (1-p)^{2^k - i} \begin{pmatrix} 2^k \\ i \end{pmatrix}$$

If $i \leq k-1$, then each node in Q_k has at least one non-faulty neighbor. Thus, the probability that Q_k has i, $i \leq k-1$, faulty nodes, and each node in Q_k has at least one non-faulty neighbor is:

$$p^i(1-p)^{2^k-i}\left(\begin{array}{c}2^k\\ \\ i\end{array}\right)$$

If $i \geq k$, then it is possible that some nodes in Q_k have no non-faulty neighbor. Moreover, in Q_k with at most $2k-3$ faulty nodes, there is at most one non-faulty node whose neighbors are all faulty. By way of contradiction, suppose there are two non-faulty nodes u and v in Q_k such that their neighbors are all faulty. Then there are at least $2k-2$ faulty nodes in Q_k because any two nodes in a hypercube can have at most two common neighbors. It contradicts the assumption that Q_k has at most $2k-3$ faulty nodes.

There are 2^k ways to choose such a non-faulty node that has no non-faulty neighbor. After that node is chosen, its k neighbors must all be faulty. Since the rest $i-k$ faulty nodes can be placed on any positions, there are $\left(\begin{array}{c}2^k-(k+1)\\ \\ i-k\end{array}\right)$ cases. Thus, if Q_k has i, $k \leq i \leq 2k-3$, faulty nodes, the probability that each node in Q_k has at least one non-faulty neighbor is:

$$p^i(1-p)^{2^k-i}\left[\left(\begin{array}{c}2^k\\ \\ i\end{array}\right)-2^k\left(\begin{array}{c}2^k-(k+1)\\ \\ i-k\end{array}\right)\right]$$

Thus, the probability that each non-faulty node in Q_k with at most $2k-3$ faulty nodes has at least one non-faulty neighbor is:

$$\sum_{i=0}^{k-1}p^i(1-p)^{2^k-i}\left(\begin{array}{c}2^k\\ \\ i\end{array}\right)+\sum_{i=k}^{2k-3}p^i(1-p)^{2^k-i}\left[\left(\begin{array}{c}2^k\\ \\ i\end{array}\right)-2^k\left(\begin{array}{c}2^k-(k+1)\\ \\ i-k\end{array}\right)\right]$$

$\square$

Lemma B.2 and Lemma B.3 give the following theorem.

Theorem B.4 *Suppose that the node failure probability in the n-cube is p. Then for any two non-faulty nodes u and v, the algorithm L_2-**Routing** constructs a path from u to v with probability at least $Pr[\mathbf{Reach}(Q_k^h)] + Pr[\mathbf{Con}(Q_k^h)] - 1$, where $Pr[\mathbf{Reach}(Q_k^h)] \geq (1-p^{k+1}(2-p)^k)(1-p^k(2-p)^{k-1})^{h-1}$ and $Pr[\mathbf{Con}(Q_k^h)] =$*

$$\sum_{i=0}^{k-1} p^i(1-p)^{2^k-i}\binom{2^k}{i} + \sum_{i=k}^{2k-3} p^i(1-p)^{2^k-i}\left[\binom{2^k}{i} - 2^k\binom{2^k-(k+1)}{i-k}\right]$$

Proof. Let h be the Hamming distance between $x_1 \cdots x_{n-k}$ and $y_1 \cdots y_{n-k}$. Under the event $\mathbf{Reach}(Q_k^h) \cap \mathbf{Con}(Q_k^h)$, the algorithm L_2-**Routing** routes successfully from u to v. The lemma holds because:

$$\begin{aligned} & Pr[\mathbf{Reach}(Q_k^h) \cap \mathbf{Con}(Q_k^h)] \\ = \;& 1 - Pr[\overline{\mathbf{Reach}(Q_k^h) \cap \mathbf{Con}(Q_k^h)}] \\ = \;& 1 - Pr[\overline{\mathbf{Reach}(Q_k^h)} \cup \overline{\mathbf{Con}(Q_k^h)}] \\ \geq \;& 1 - Pr[\overline{\mathbf{Reach}(Q_k^h)}] - Pr[\overline{\mathbf{Con}(Q_k^h)}] \\ = \;& Pr[\mathbf{Reach}(Q_k^h)] + Pr[\mathbf{Con}(Q_k^h)] - 1 \end{aligned}$$

□

Specifically, for given n, k, and p, $Pr[\mathbf{Reach}(Q_k^h)] + Pr[\mathbf{Con}(Q_k^h)] - 1$ gives a lower bound on the success probability for the L_2-**Routing** when $h = n-k$. Let us set $k = 5$ and the node failure probability to $p = 6\%$ and calculate the success probability of the algorithm L_2-**Routing** for different dimension of n-cube. The results are given in Table II. The value of node failure probability p is chosen so that we can make the lower bound of success probability over 99%. From this table, it shows that when the node failure probability is less than or equal to 6%, the L_2-**Routing** algorithm routes successfully with probability over 99.9%.

Table II. Success probability of the algorithm L_2-**Routing**

Dimension of Hypercube Q_n	Success Probability $p = 6\%$
10	$\geq .9995$
15	$\geq .9994$
20	$\geq .9993$
25	$\geq .9992$
30	$\geq .9992$
35	$\geq .9991$
40	$\geq .9991$

C. L_2-Parallel-Routing

In L_2-**Routing**, we use one single path through $h+1$ k-subcubes for finding a routing path from u to v. If we can use node-disjoint paths from u to v, then we can improve the success probability to find a path between them. Suppose the source node $u = x_1x_2\cdots x_{n-k}**$ is in the k-subcube Q_k, and the destination node $v = y_1y_2\cdots y_{n-k}**$ is in the k-subcube Q'_k. Also, suppose that the Hamming distance between the substrings $x_1x_2\cdots x_{n-k}$ and $y_1y_2\cdots y_{n-k}$ is h. Then we can pair a neighbor u_i of u and a neighbor v_j of v by **Prematch-I** which is introduced in Chapter III. For the paired neighbors u_i and v_j, if u_i and v_j are non-faulty, then we can use L_2-**Routing** between u_i and v_j with a slight modification. That is, if $1 \leq i \leq h$, then we traverse through $h-1$ k-subcubes by converting the bits x_b of u_i into y_b of v_j in the order of $b = i, i+1, \ldots, h, 1, \ldots, i-2, j$. Also, if $h+1 \leq i \leq n-k$, then we traverse through $h+1$ k-subcubes by converting the bits x_b of u_i into y_b of v_j in the order

Algorithm. L_2**-Parallel-Routing**

Input: two non-faulty nodes $u = x_1x_2 \ldots x_n$ and $v = y_1y_2 \ldots y_n$ in the n-cube network Q_n.

Output: a fault-free path in Q_n from u to v.

1. let u is in a k-subcube Q_k and v is in a k-subcube Q'_k;
2. let h be the Hamming distance between $x_1x_2 \cdots x_{n-k}$ and $y_1y_2 \cdots y_{n-k}$;
3. **for** each pair (u_i, v_j) given by **Prematch-I** such that u_i and v_j are non-faulty **do**

3.1 **case 1.** $1 \leq i \leq h$
use the L_2**-Routing** along k-subcubes $Q_k^1, Q_k^2, \ldots, Q_k^{h-1}$ in the order of dimension $i, i+1, \ldots, h, 1, \ldots, i-2, j$ where
$u_i \in Q_k^1 = u_1 \cdots \overline{u_i} \cdots u_{n-k} **$ and
$v_j \in Q_k^{h-1} = v_1 \cdots \overline{v_j} \cdots v_{n-k} **$;

3.2 **case 2.** $h+1 \leq i \leq n-k$
use the L_2**-Routing** along k-subcubes $Q_k^1, Q_k^2, \ldots, Q_k^{h+1}$ in the order of dimension $i, 1, 2, \ldots, h, j$ where
$u_i \in Q_k^1 = u_1 \cdots \overline{u_i} \cdots u_{n-k} **$ and
$v_j \in Q_k^{h+1} = v_1 \cdots \overline{v_j} \cdots v_{n-k} **$;

3.3 **if** L_2**-Routing** return a fault-free path P between u_i and v_j **then**
extend P into the path from u to v;
return the path P;

4. return NULL; ('routing fail')

Fig. 17. The algorithm L_2-Parallel-Routing

of $b = i, 1, 2, \ldots, h, j$. The algorithm called L_2**-Parallel-Routing** is presented in Fig. 17.

If we regard each k-subcube as a single node, then the Q_n is regarded as an $n-k$ dimensional hypercube Q_{n-k}. From Lemma C.1, for two pairs (u_x, u_y) and (u_s, v_t) given by **Prematch-I**, k-subcubes traversed between u_x and v_y and k-subcubes traversed between u_s and v_t are disjoint. That is, we perform the routing along the k-subcubes corresponding to each path in the set of node-disjoint paths between u and v.

Lemma C.1 *Suppose that the algorithm* L_2**-Parallel-Routing** *routes successfully from* u *to* v*, then in time* $O(kn(h+2^k))$*,* L_2**-Parallel-Routing** *finds a fault-free* *path P of length bounded by $2h+k+4$, where h is the Hamming distance between $x_1 \cdots x_{n-k}$ and $y_1 \cdots y_{n-k}$.*

PROOF. For a pair (u_i, v_j) given by **Prematch-I**, suppose both u_i and v_j are non-faulty. If $1 \le i \le h$, then step 3.1 of the algorithm L_2**-Parallel-Routing** traverses through $h-1$ k-subcubes $Q_k^1, \ldots, Q_k^{h-1}$, where u_i is in Q_k^1 and v_j is in Q_k^{h-1}. If $h+1 \le i \le n-k$, then step 3.2 of the algorithm L_2**-Parallel-Routing** traverses through $h+1$ k-subcubes $Q_k^1, \ldots, Q_k^{h+1}$, where u_i is in Q_k^1 and v_j is in Q_k^{h+1}. Since we move from a node in a k-subcube Q_k^i to a node in its neighboring k-subcube Q_k^{i+1} within at most 2 hops, the length of the path P between u_i and v_j constructed by the algorithm L_2**-Routing** is bounded by $2h+k+2$. Step 3.3 of the algorithm L_2**-Parallel-Routing** extends P to the path from u to v by adding two edges $< u, u_i >$ and $< v_j, v >$. Thus, if the algorithm L_2**-Parallel-Routing** finds a fault-free path P from the node u to the node v, then the length of the path P is bounded by $2h+k+4$.

From Lemma B.1, steps 3.1 and 3.2 of the algorithm L_2**-Parallel-Routing** takes $O(hk+k2^k)$. Since step 3 of the algorithm L_2**-Parallel-Routing** will execute at most $n-k$ times, the running time of the algorithm L_2**-Parallel-Routing** is bounded by $O((n-k)(hk+k2^k)) = O(kn(h+2^k))$. □

We compute the success probability of the algorithm L_2**-Parallel-Routing** in the following theorem:

Theorem C.2 *Suppose that the node failure probability in the n-cube is p. Then for any two non-faulty nodes u and v, the algorithm* L_2**-Parallel-Routing** *constructs a path from u to v with probability at least* $1-(1-(1-p)^2(Pr[\mathbf{Reach}(Q_k^h)] + Pr[\mathbf{Con}(Q_k^h)]-1))^{n-k}$.

Proof. For $1 \le i \le n-k$, if both u_i and v_j are non-faulty, and u_i can reach v_j by the algorithm L_2**-Routing**, then u can reach v_j in one hop, and v_j can reach v in one hop. That is, the node u can reach the node v by going through the path between u_i and v_j. From Theorem B.4, with a probability of at least $Pr[\mathbf{Reach}(Q_k^h)] + Pr[\mathbf{Con}(Q_k^h)] - 1$, the algorithm L_2**-Routing** constructs a path from u_i to v_j. Step 3 of the algorithm L_2**-Parallel-Routing** will execute at most $n-k$ times. During each execution of the loop, the probability that the algorithm L_2**-Parallel-Routing** cannot return a fault-free path is at most $1-(1-p)^2(Pr[\mathbf{Reach}(Q_k^h)] + Pr[\mathbf{Con}(Q_k^h)] - 1)$. Since k-subcubes traversed in each iteration are disjoint, the failure probability that for each pair (u_i, v_j) given by **Prematch-I**, u_i cannot reach v_j is independent. Thus, the probability that u cannot reach v in the algorithm L_2**-Parallel-Routing** is at most $(1-(1-p)^2(Pr[\mathbf{Reach}(Q_k^h)] + Pr[\mathbf{Con}(Q_k^h)] - 1))^{n-k}$. Therefore, the probability that the algorithm L_2**-Parallel-Routing** constructs a path from u to v is at least $1-(1-(1-p)^2(Pr[\mathbf{Reach}(Q_k^h)] + Pr[\mathbf{Con}(Q_k^h)] - 1))^{n-k}$. □

For given n, k, and p, $1-(1-(1-p)^2(Pr[\mathbf{Reach}(Q_k^h)] + Pr[\mathbf{Con}(Q_k^h)] - 1))^{n-k}$ gives a lower bound on the success probability for the L_2**-Parallel-Routing** when $h = n-k$. Table III shows the success probability of the algorithm L_2**-Parallel-Routing** for different dimensions of n-cube when $k = 4$, and $p = 20\%$. From this table, it shows that when failure probability is lower than or equal to 20%, the algorithm L_2**-Parallel-Routing** routes successfully with probability over 99%. When the dimension of the hypercube is getting larger, the success probability is very close to 1. Compared to L_2**-Routing**, the algorithm L_2**-Parallel-Routing** can tolerate much larger node failure probability p.

Table III. Success probability of the algorithm L_2-**Parallel-Routing**

Dimension of Hypercube Q_n	Success Probability $p = 20\%$
10	$\geq .9919$
15	$\geq .9997$
20	$\geq .99998$
25	$\geq .9999985$
30	$\geq .9999998$
35	$\geq .99999996$
40	$\geq .99999999$

D. Chapter Summary

We have presented two routing algorithms, L_2-**Routing** and L_2-**Parallel-Routing** that construct a fault-free path between any two given non-faulty nodes in hypercube networks. Without considering the global connectivity of the whole network, the suggested algorithms construct a fault-free path between any two given nodes in hypercubes with very high probability. When the failure probability for each node is 6%, for any hypercube whose dimension is not larger than 40, the probability that our algorithm L_2-**Routing** can find a fault-free path is over 99.9%. Suppose that we are given a source node $u = x_1x_2 \cdots x_n$, and a destination node $v = y_1y_2 \cdots y_n$, and that the Hamming distance between the strings $x_1x_2 \cdots x_{n-k}$ and $y_1y_2 \cdots y_{n-k}$ is h. Suppose L_2-**Routing** routes from u to v through 5-subcubes $Q_5^0, Q_5^1, \ldots, Q_5^h$, then in time $O(h+c)$, L_2-**Routing** finds a fault-free path of length bounded by $2h+7$. Since h is bounded by $O(n)$, the length of the routing path constructed by L_2-**Routing** is bounded by $O(n)$. We further investigated our routing algorithm to allow a larger failure probability. We applied a method suggested in Chapter III to L_2**-Parallel-Routing**. By using disjoint paths from u to v, when the failure probability for each node is no more than 20%, the probability that our algorithm L_2**-Parallel-Routing** can find a fault-path is over 99%.

CHAPTER V

CONCLUSIONS

A. Thesis Summary

Strong fault tolerance is a natural extension of the study of network fault tolerance and parallel routing. In particular, it is the study of fault tolerance on large size networks with faulty nodes. In Chapter II and Chapter III, we demonstrated that the popular interconnection networks, such as the star networks and the hypercube networks, are strongly fault tolerant. We presented an algorithm of running time $O(n^2)$ that for two given non-faulty nodes u and v, constructs the maximum number (i.e., $\min\{deg_f(u), deg_f(v)\}$) of node-disjoint fault-free paths from u to v such that the length of the paths is bounded by $dist(u,v)+8$ for the star networks and bounded by $dist(u,v)+4$ for the hypercube networks. The time complexity of our algorithm is optimal since each path from u to v in the network S_n or Q_n may have a length as large as $\Theta(n)$, and there can be as many as $\Theta(n)$ node-disjoint paths from u to v. Thus, even printing these paths should take time $O(n^2)$. We have shown that the length of the paths constructed by our algorithm for the star networks is almost optimal. For the n-cube network Q_n, the length of the paths constructed by our algorithm is bounded by $dist(u,v)+4$. It is not difficult to see that this is the best possible, since there are node pairs u and v in Q_n with $n-2$ faulty nodes, for which any group of $\min\{deg_f(u), deg_f(v)\}$ parallel paths from u to v contains at least one path of length at least $dist(u,v)+4$.

In chapter IV, we investigated the fault tolerance of hypercube networks by using a probability model. In this research, we focused on developing routing algorithms that, for two given nodes in n-cubes, can find a fault-free path with very high probability while keeping the length of the path bounded by $O(n)$. We assume that each node in hypercube networks has an independent failure probability. Under this model, we analyzed the success probability that algorithms can return a fault-free path. Without considering the global connectivity of the whole network, the suggested routing algorithms find a fault-free path with very high success probability. Compared to the previous scheme proposed in [9], our schemes would be more attractive for users who want to find a fault-free routing path without considering the global connectivity of the whole network.

B. Future Research

The hypercube networks and the star networks are the first two classes of networks whose strong fault tolerance have been proved. For star networks, the strong fault tolerance was proved based on the orthogonal partition of the star networks, while for hypercube networks, the strong fault tolerance was proved by careful pre-matching of the neighbors of the source and destination nodes. Strong fault tolerance for networks with bounded degree, such as ring networks, mesh networks, and butterfly networks, are relatively easier. On the other hand, strong fault tolerance for unbounded degree networks, such as networks based on Cayley graphs, seems much more difficult. It will be interesting to study the strong fault tolerance of other hierarchical networks with unbounded degree.

The probability model used for the hypercube networks can be applied to other hierarchical network structures such as a variety of hypercube variations. Specifically, our model can be easily applied to k-ary n-dimensional hypercube networks which are general forms of binary hypercube networks. Also, studying the probability of fault tolerance for networks with degree bounded by a small constant would be interesting. In this thesis, we assume that each node in the networks has a uniform and independent failure probability. In reality, nodes may have different failure probabilities which make the distribution of node failure probability nonuniform. In addition, nodes may be related and fail at the same time, so that node failures may not be independent. Our study can be extended to handle these probability models.

REFERENCES

[1] S. B. Akers, D. Harel, and B. Krishnamurthy, "The Star Graph: An Attractive Alternative to The n-Cube," *Proc. Intl. Conf. of Parallel Processing*, pp. 393-400, 1987.

[2] S. B. Akers and B. Krishnamurthy, "A Group Theoretic Model for Symmetric Interconnection Networks," *IEEE Trans. on Computers*, vol. 38, pp. 555-565, 1989.

[3] S. B. Akers and B. Krishnamurthy, "The Fault Tolerance of Star Graphs," *Proc. 2nd International Conference on Supercomputing*, San Francisco, CA, pp. 270-276, 1987.

[4] G. Birkhoff and S. MacLane, *A Survey of Modern Algebra*, New York: The Macmillan Company, 1965.

[5] A. Broder, D. Dolev, M. J. Fischer, and B. Simons, "Efficient Fault-Tolerant Routings in Networks," *Information and Computation*, vol. 75, no. 1, pp. 52-64, 1987.

[6] N. Bagherzadeh, N. Nassif, and S. Latifi, "A Routing and Broadcasting Scheme on Faulty Star Graphs," *IEEE Trans. on Computers*, vol. 42, pp. 1398-1403, 1993.

[7] C. C. Chen and J. Chen, "Optimal Parallel Routing in Star Networks," *IEEE Trans. on Computers*, vol. 46, pp. 1293-1303, 1997.

[8] C. C. Chen and J. Chen, "Nearly Optimal One-to-Many Parallel Routing in Star Networks," *IEEE Trans. Parallel, Distrib. Syst.*, vol. 8, pp. 1196-1202, 1997.

[9] J. Chen, I. A. Kanj, and G. Wang, “Hypercube Network Fault Tolerance: A Probabilistic Approach,” *Proc. International Conference on Parallel Processing*, Vancouver, B.C., Canada, pp. 65-72, 2002.

[10] J. Chen, G. Wang, and S. Chen, “Routing in Hypercube Networks with A Constant Fraction of Faulty Nodes,” *Proc. 4th International Conference on Algorithms and Architecture for Parallel Processing*, pp. 605-616, 2000.

[11] J. Chen, G. Wang, and S. Chen, “Locally Subcube-Connected Hypercube Networks: Theoretical Analysis and Experimental Results,” *IEEE Trans. on Computers*, vol. 51, no. 5, pp. 530-540, 2002.

[12] G. -M. Chiu and S. -P. Wu, “A Fault-Tolerant Routing Strategy in Hypercube Multicomputers,” *IEEE Trans. Computers* vol. 45, pp. 143-154, 1996.

[13] M. -S. Chen and K. G. Shin, “Adaptive Fault-Tolerant Routing in Hypercube Multicomputers,” *IEEE Trans. Computers* vol. 39, pp. 1406-1416, 1990.

[14] M. -S. Chen and K. G. Shin, “Depth-First Search Approach for Fault-Tolerant Routing in Hypercube Multicomputers,” *IEEE Trans. Parallel and Distributed Systems*, vol. 1, no. 2, pp. 152-159, 1990.

[15] H. Choi, S. Subramaniam, and H. -A. Choi, “Loopback Recovery from Double-Link Failures in WDM Optical Mesh Networks,” *IEEE/ACM Trans. Networks*, to appear

[16] H. Choi, S. Subramaniam, and H. -A. Choi, “On Double-Link Failure Recovery in WDM Optical Networks,” *Proc. INFOCOM*, New York, 2002.

[17] K. Day and A. Tripathi, “A Comparative Study of Topological Properties of Hypercubes and Star Graphs,” *IEEE Trans. Parallel, Distrib. Syst.* , vol. 5, pp.

31-38, 1994.

[18] M. Dietzfelbinger, S. Madhavapeddy, and I. H. Sudborough, "Three Disjoint Path Paradigms in Star Networks," *Proc. 3nd IEEE Symposium on Parallel and Distributed Processing*, Dallas, TX, pp. 400-406, 1991.

[19] D. Dolev, J. Y. Halpern, B. Simons, and H. R. Strong, "A New Look at Fault-Tolerant Network Routing," *Information and Computation*, vol. 72, no. 3, pp. 180-196, 1987.

[20] P. Fraigniaud, "Asymptotically Optimal Broadcasting and Gossiping in Faulty Hypercube Multicomputers," *IEEE Trans. Computers* vol. 41, pp. 1410-1419, 1992.

[21] S. Fujita, "A Fault-Tolerant Broadcast Scheme in the Star Graph under the Single-Port, Half-Duplex Communication Model," *IEEE Trans. on Computers*, vol. 48, pp. 1123-1126, 1999.

[22] D. Goyal and J. Caffery, Jr. "Partitioning Avoidance in Mobile Ad Hoc Networks Using Network Survivability Concepts," *Proc. 7th IEEE Symposium on Computers and Communications*, Taormina, Italy, 2002.

[23] L. Gargano, A. Rescigno, and U. Vaccaro, "Minimum Time Broadcast in Faulty Star Networks," *Discrete Applied Mathematics*, vol. 83, pp. 97-119, 1998.

[24] Q. -P. Gu and S. Peng, "Optimal Algorithms for Node-to-Node Fault Tolerant Routing in Hypercubes," *The Computer Journal* vol. 39, pp. 626-629, 1996.

[25] Q.-P. Gu and S. Peng, "Fault Tolerant Routing in Hypercubes and Star Graphs," *Parallel Processing Letters*, vol. 6, pp. 127-136, 1996.

[26] Q.-P. Gu and S. Peng, "An Efficient Algorithm for k-Pairwise Disjoint Paths in Star Graphs," *Information Processing Letters*, vol. 67, pp. 283-287, 1998.

[27] Q.-P. Gu and S. Peng, "Unicast in Hypercubes with Large Number of Faulty Nodes," *IEEE Trans. Parallel and Distributed Systems*, vol. 10, pp. 964-975, 1999.

[28] Q. -P. Gu and S. Peng, "An Efficient Algorithm for the k-Pairwise Disjoint Paths Problem in Hypercube," *J. Parallel and Distributed Computing*, vol. 60, pp. 764-774, 2000.

[29] P. Hall, "On Representatives of Subsets," *J. London Math. Soc.*, vol. 10, pp. 26-30, 1935.

[30] J. Jwo, S. Lakshmivarahan, and S. K. Dhall, "Characterization of Node Disjoint (Parallel) Path in Star Graphs," *Proc. 5th Intl. Parallel Processing Symp.*, pp. 404-409, 1991.

[31] K. Li, "Topological Characteristics of Random Multihop Wireless Networks," *Proc. 23rd Conf. Distributed Computing Systems Workshops* Providence, Rhode Island, 2003.

[32] M. S. Krishnamoorthy and B. Krishnamurthy, "Fault Diameter of Interconnection Networks," *Comput. Math. Appl.*, vol. 13, pp. 577-582, 1987.

[33] S. Latifi, "Combinatorial Analysis of the Fault-Diameter of the n-Cube," *IEEE Trans. Computers* vol. 42, pp. 27-33, 1993.

[34] S. Latifi, "On the Fault-Diameter of the Star Graph," *Information Processing Letters*, vol. 46, pp. 143-150, 1993.

[35] S. Latifi, H. Ko, and P. K. Srimani, “Node-to-Set Vertex Disjoint Paths in Hypercube Networks,” *Technical Report CS-98-107*, Colorado State University, 1998.

[36] N. W. Lo, B. S. Carlson, and D. L. Tao, “Fault Tolerant Algorithms for Broadcasting on the Star Graph Network,” *IEEE Trans. on Computers*, vol. 46, pp. 1357-1362, 1997.

[37] S. Lee and K. Shin, “Interleaved All-to-All Reliable Broadcast on Meshes and Hypercubes,” *Proc. Int’l Conf. Parallel Processing*, pp.III-110-113, 1990.

[38] T. C. Lee and J. P. Hayes, “Routing and Broadcasting in Faulty Hypercube Computers,” *Proc. third Conf. Hypercube Concurrent Computers and Applications*, pp. 625-630, 1988.

[39] X. Lin and I. Stojmenovic, “Location-based Localized Alternate, Disjoint and Multi-path Routing Algorithms for Wireless Networks,” *J. Parallel and Distributed Computing*, vo. 63, pp. 22-32, 2003.

[40] S. Madhavapeddy and I. H. Sudborough, “A Topological Property of Hypercubes: Node Disjoint Paths,” *2nd IEEE Symposium on Parallel and Distributed Processing*, pp. 532-539, 1990.

[41] K. Menger, “Zur Allgemeinen Kurventheorie,” *Fund. Math.*, vol. 10, pp. 96-115, 1927.

[42] G. D. Marco and U. Vaccaro, “Broadcasting in Hypercubes and Star Graphs with Dynamic Faults,” *Information Processing Letters*, vol. 66, pp. 321-326, 1998.

[43] A. Nasipuri, R. Castaneda, S. R. Das, “On-demand Multipath Routing for Mobile Ad Hoc Networks,” *Proc. INFOCOM*, New York, 1999.

[44] S. Park and B. Bose, "All-to-All Broadcasting in Faulty Hypercubes," *IEEE Trans. on computers*, vol. 46, pp. 749-755, 1997.

[45] M. Pearlman, Z. Haas, P. Sholander, and S. S. Tabrizi, "Alternate Path Routing in Mobile Ad Hoc Networks," *Proc. IEEE MILCOM*, Los Angeles, CA, 2002.

[46] K. Paul, R. Roychoudhuri, and S. Bandyopadhyay, "Survivability Analysis of Ad Hoc Wireless Network Architecture," *Proc. Mobile and Wireless Communication Networks-MWCN*, LNCS 1818, pp. 31-46, 2000.

[47] D. Peleg and B. Simons, "On Fault Tolerant Routings in General Networks," *Information and Computation*, vol. 74, no. 1, pp. 33-49, 1987.

[48] M. O. Rabin, "Efficient Dispersal of Information for Security, Load Balancing, and Fault Tolerance," *Journal of ACM*, vol. 36, pp. 335-348, 1989.

[49] A. A. Rescigno, "Fault-Tolerant Parallel Communication in the Star Network," *Parallel Processing Letters*, vol. 7, pp. 57-68, 1997.

[50] P. Ramanathan and K. Shin, "Reliable Broadcast in Hypercube Multicomputers," *IEEE Trans. Computers* vol. 37, pp. 1654-1657, 1988.

[51] A. A. Rescigno and U. Vaccaro, "Highly Fault-Tolerant Routing in the Star and Hypercube Interconnection Networks," *Parallel Processing Letters*, vol. 8, pp. 221-230, 1998.

[52] Y. Saad and M. H. Schultz, "Topological Properties of Hypercubes," *IEEE Transactions on Computers*, vol. 37, pp. 867-872, 1988.

[53] C. L. Seitz, "The Cosmic Cube," *Communication ACM*, vol. 28, pp. 23-23, 1985.

[54] S. Sur and P. K. Srimani, "Topological Properties of Star Graphs," *Computers Math. Applic.*, vol. 25, pp. 87-98, 1993.

[55] S. Vutukury and J. J. Garcia-Luna-Aceves, "MDVA: A Distance-vector Multipath Routing Protocol," *Proc. INFOCOM*, Anchorage, AK, 2001.

VITA

Eunseuk Oh was born on March, 29, 1971 in Chuncheon, Korea. She received her B.S. degree in Computer Science from Hallym University and her M.S. degree in Computer Science from Ewha Womans University in 1994 and 1996, respectively. Since September of 1997, she has been with the Computer Science Department at Texas A&M University pursuing her Ph.D. under the supervision of Dr. Jianer Chen. Her areas of interests include fault tolerant routing in interconnection networks, design of interconnection networks, network survivability, graph algorithms, and combinatorics. Her permanent mailing address is 2310 Westcreek Lane, College Station, Texas, 77845, USA.

The typist for this thesis was Eunseuk Oh.