AN ABSTRACT OF THE DISSERTATION OF

Omar I. Alsaleh for the degree of Doctor of Philosophy in Computer Science presented on December 5, 2013.

Title: One-to-Many Node Disjoint Paths Routing in Generalized Hypercube, Dense Gaussian, and Hexagonal Mesh Networks

Abstract approved: ______________________________

Bella Bose Bechir Hamdaoui

Routing from a single source node to multiple destination nodes using node disjoint paths (NDP) has many important applications in parallel systems. For example, if a source node wants to send distinct messages to distinct destination nodes, then the one-to-many NDP routing is useful.

Unlike parallel systems with shared-memory, each node in most of the current parallel systems is a standalone processing unit with a processor and memory. The nodes communicate with each other by passing messages using a standard message passing mechanism such as the Message Passing Interface (MPI). The probability of failure in delivering the messages between the nodes directly affects the computing performance. This probability increases as the number of nodes increases. Therefore, it is critical to find a set of mutually node disjoint paths in order to establish communication routes under such faulty environment. Moreover, the one-to-many NDP routing increases the throughput of the networks.

In this work, we provide some novel and efficient routing algorithms that construct a set of NDP from a single source node to the maximum number of destination nodes in three promising interconnection networks. They are Generalized Hypercube, dense Gaussian, and Hexagonal Mesh networks.

In Chapter two, two efficient algorithms that construct a set of NDP from a single source node to the maximum number of destination nodes in Generalized Hypercube networks are given. Also, the lower and upper bounds of the path length from the source node to any destination node are derived. Finally, some simulations of the algorithms are performed and the results show that in most cases the maximum path length is equal to the shortest distance plus one.

In Chapter three and Chapter four, efficient constant time complexity algorithms that construct a set of one-to-many NDP in dense Gaussian and Hexagonal Mesh networks are given, respectively. For the dense Gaussian network, the lower and upper bounds of the sum of the NDP lengths are derived. Also, via execution of the algorithm, it is shown that on average the sum of the lengths of NDP given by the algorithm is only about 10% more than the sum of the lengths of the shortest paths.

One-to-Many Node Disjoint Paths Routing in Generalized Hypercube, Dense Gaussian, and Hexagonal Mesh Networks

by

Omar I. Alsaleh

A DISSERTATION

submitted to

Oregon State University

in partial fulfillment of
the requirements for the
degree of

Doctor of Philosophy

Presented December 5, 2013
Commencement June 2014

Doctor of Philosophy dissertation of Omar I. Alsaleh presented on December 5, 2013.

APPROVED:

Major Professor, representing Computer Science

Director of the School of Electrical Engineering and Computer Science

---

Dean of the Graduate School

I understand that my dissertation will become part of the permanent collection of Oregon State University libraries. My signature below authorizes release of my dissertation to any reader upon request.

Omar I. Alsaleh, Author

ACKNOWLEDGEMENTS

Earning the doctoral degree has been a very long journey. Many people helped me to reach the end of it. I would like to thank them all especially the following:

- My parents (Ibrahim Alsaleh and Fawziah Alhassoun) for planting the value of education in me and taking good care of their seed over the years. I give them this degree as their harvest and hope it makes them happy and satisfy.

- My wife (Bushra Alsuliman) for supporting me and making lots of sacrifices including living oversees away from her family. Without my wife, I would never have been completed this work.

- My precious kids (Abdullah and Jenna) for spreading happiness around my life.

- My only brother (Abdullah Alsaleh) and sisters for believing in me and looking at me as a good role model.

- My advisors (Dr. Bella Bose and Dr. Bechir Hamdaoui) for not only unconditionally helping me to complete this work, but also for teaching me how to conduct a high quality academic research. I cannot thank them enough. They are good friends of mine for life.

- My doctoral committee members (Dr. Huaping Liu, Dr. Eduardo Cotilla-Sanchez, and Dr. Harold Parks) for cutting from their valuable time to serve in my committee.

- Dr. Prasad Tadepalli and Dr. Alan Fern for supporting and helping me at the beginning of my doctoral study.
- The School of Electrical Engineering and Computer Science in Oregon State University for providing high quality education.
- College of Computer and Information Sciences in King Saud University, Saudi Arabia, for the full paid scholarship.
- My advisor (Dr. Abdulkader Alfantookh, Deputy Minister for Planning and Information Ministry of Higher Education in the Ministry of Higher Education, Saudi Arabia) who supervised my bachelor's and master's theses for building my academic base that I stood on it to earn a doctoral degree.
- My teacher and friend (Dr. Abdullah Aldhelaan, Vice Dean for Graduate Studies and Scientific Research in King Saud University, Saudi Arabia) for the the priceless support before and during the doctoral program.
- My fellow graduate students in the School of Electrical Engineering and Computer Science for all kind of cooperation and supporting each other.
- My best friends (Haitham Alsaqer and Abdulrahman Alkhenifer) who supported me during my master's studies and become the best friends of mine.

Those are what I can remember and I am sure I forgot a lot of other people. I thank them all from the bottom of my heart.

TABLE OF CONTENTS

TABLE OF CONTENTS (Continued)

LIST OF FIGURES

LIST OF FIGURES (Continued)

LIST OF FIGURES (Continued)

Figure Page

LIST OF FIGURES (Continued)

LIST OF TABLES

LIST OF ALGORITHMS

Algorithm Page

Chapter 1: Introduction

1.1 Overview

Since the switching speed of VLSI systems is approaching the maximum limit, parallel systems play an important role in improving the system performance by exploiting the inherent parallelism in problems. In the last decade, supercomputers with thousands of nodes have been built, such as: the Cray Jaguar [6], the IBM BlueGene [2], etc. The nodes are linked to each other to form an interconnection network.

Achieving high computing performance critically depends on the interconnection networks. Designers of the interconnection networks seek desirable attributes such as low node degree, small diameter, and strong fault tolerance to maximize the computing performance [13, 14, 24, 34, 41]. As a result, many different topologies have been extensively investigated in the literature in order to find which ones yield the best computing performance [1, 5, 12–14, 24, 27, 34, 41].

Unlike the shared-memory parallel systems, each node in most of the current parallel systems is a standalone processing unit with a processor and memory. The nodes communicate with each other by passing messages using a standard message passing mechanism such as the Message Passing Interface (MPI) [2]. The probability of failure in delivering the messages between the nodes directly affects the computing performance. This probability increases as the number of nodes increases. Therefore, it is critical to find a set of mutually node disjoint paths

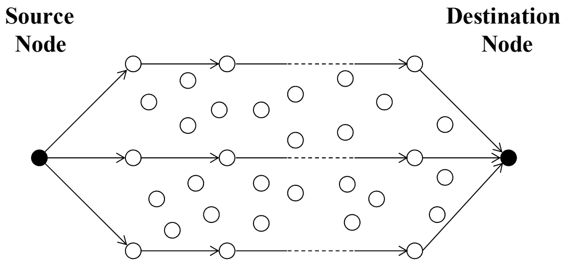


Figure 1.1: One-to-one NDP

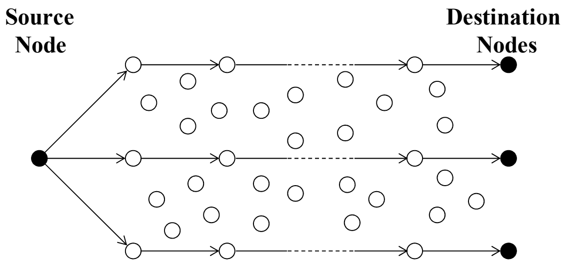


Figure 1.2: One-to-many NDP

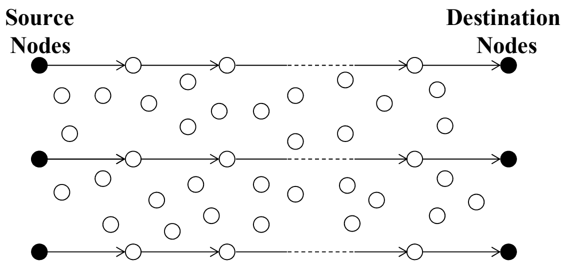


Figure 1.3: Many-to-many NDP

(NDP) in order to establish communication routes under such faulty environment. Finding this set is fundamental and essential for ensuring fault tolerance in parallel systems. It is used to connect: a source node to a destination node (one-to-one, see Figure 1.1), a source node to a set of destination nodes (one-to-many, see Figure 1.2), or a set of source nodes to a set of destination nodes (many-to-many, see Figure 1.3).

The one-to-many NDP routing problem is described as follows: given a source node s, a set of distinct destination nodes $T = \{t_1, t_2, \ldots, t_\ell\}$, where $s \notin T$ and ℓ

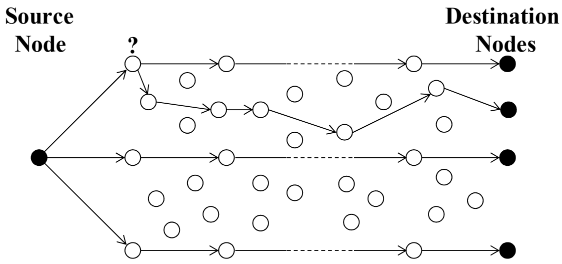


Figure 1.4: The source degree equals the maximum number of destination nodes

is the source degree, construct a set of ℓ NDP such that: 1) each path connects the source node s to one of the destination nodes $t_i \in T$, $i \in \{1, 2, \ldots, \ell\}$, and 2) the only common node between any pair of these paths is the source node s. The source's degree ℓ equals the maximum number of destination nodes in any regular graph because the solution does not exist if the number of destination nodes is greater than the source's degree (see Figure 1.4). In this work, we provide novel and efficient routing algorithms that construct a set of one-to-many NDP from a source node to the maximum number of destination nodes in Generalized Hypercube, dense Gaussian, and Hexagonal Mesh networks.

Unlike the existing types of hypercube, the Generalized Hypercube supports any number of nodes. However, it possesses a small average message distance and a low traffic density, thereby making it highly fault tolerant. A two-dimensional Generalized Hypercube which employs optical fibers for wires has been built [43].

The dense Gaussian networks have significant topological advantages over torus networks in terms of diameter [31]. For example, there is a dense Gaussian network with 400 nodes and diameter 14, whereas, any 2D toroidal network with 400 nodes will have a diameter of at least 20. So, compared to torus networks, dense Gaussian networks can accommodate more nodes with less communication latency and at the same time maintaining a regular grid-like structure. This makes dense Gaussian

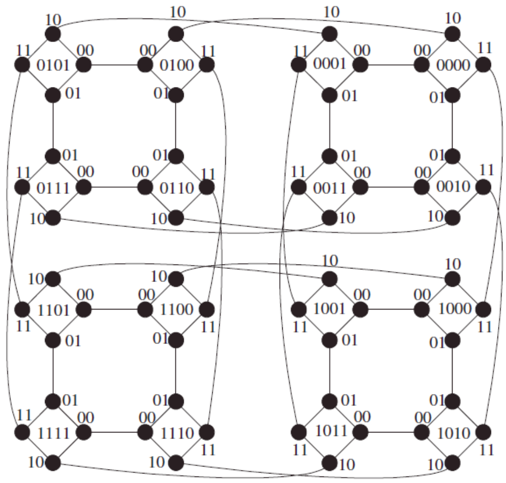


Figure 1.5: Hierarchical Hypercube [39]

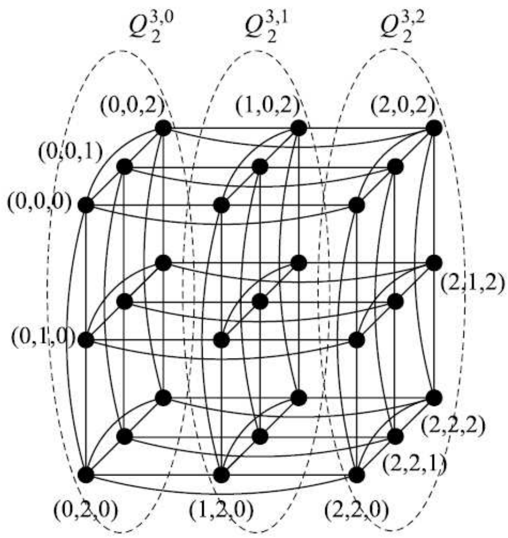


Figure 1.6: 3-ary 3-cube [7, 37]

networks attractive.

1.2 Related Work

The node disjoint paths (NDP) problems have been studied for different interconnection networks. The following related works are some examples [8–11, 20–

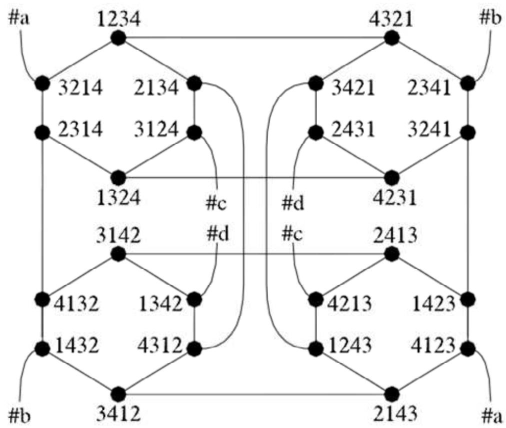


Figure 1.7: 4-pancake graph [23]

23, 25, 26, 28–30, 35, 37–40]:

- **One-to-One NDP:** This problem has been solved for the following interconnection networks: Hierarchical Hypercube [39] (Figure 1.5), k-ary n-cube [7, 37] (Figure 1.6), Hypercube [38], and (n, k)-Star [40].

- **Many-to-Many NDP:** This problem has been solved for the following interconnection networks: Hierarchical Hypercube [8], Metacube [35], Dual-Cubes [21], and Hypercube [30].

- **One-to-Many NDP:** This problem has been solved for the following interconnection networks: Hierarchical Hypercube [10], Dual-Cubes [20], Metacube [9], Folded Hypercube [26], Biswapped [28], Hypercube in optimal time [25], Hyper-Star [29], k-ary n-cube [16], Rotator graphs [22], and pancake graphs [23] (Figure 1.7).

Unlike those previous works, we solve the problem of routing from a single source node to the maximum number of destination nodes (one-to-many) in Generalized Hypercube, dense Gaussian, and Hexagonal Mesh networks using NDP.

1.3 Organization of this Dissertation

The organization of this dissertation is as follows: Chapter 2, Chapter 3, and Chapter 4 provide the one-to-many NDP in Generalized Hypercube, dense Gaussian, and Hexagonal Mesh networks, respectively. Each chapter starts with some explanation of the topology, and then the routing algorithm. Chapter 5 concludes this work and provides some possible future work.

Chapter 2: One-to-Many Node Disjoint Paths Routing in Generalized Hypercube Networks

In this chapter, two efficient algorithms that construct a set of node disjoint paths (NDP) from a single source node to the maximum number of destination nodes in Generalized Hypercube (GH) networks are given. Then, it is proved that these algorithms always return a solution. Also, the lower and upper bounds of the path length from the source node to any destination node are derived. Finally, some simulation of the algorithms are performed and the results show that most of the time the upper bound is equal to the shortest distance plus one.

Unlike the existing types of hypercube, the GH supports any number of nodes. However, it possesses a small average message distance and a low traffic density, thereby making it highly fault tolerant. A two dimensional GH which employs optical fibers for wires has been built [43].

The rest of this chapter is organized as follows: Section 2.1 recalls several preliminaries about the GH, Section 2.2 describes the proposed routing algorithm for two-dimensional GH, Section 2.3 describes the proposed algorithm for n-dimensional GH, Section 2.4 shows the simulation results, and finally Section 2.5 concludes this chapter.

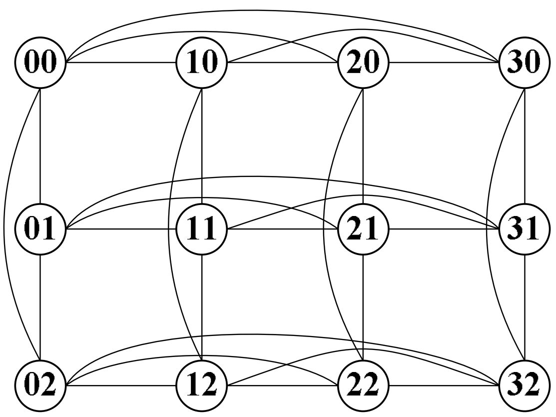


Figure 2.1: The Generalized Hypercube $Q^2_{4,3}$

2.1 Generalized Hypercube Networks Preliminaries

Unlike the Boolean n-cube structure which is an interconnection of 2^n nodes, a Generalize Hypercube (GH) denoted by $Q^n_{k_{n-1},\ldots,k_1,k_0}$ supports any number of nodes N such that $N = \prod_{i=0}^{n-1} k_i$, where n is the number of dimensions, and k_i is the number of nodes along the i-th dimension. Figure 2.1 shows an example of a two-dimensional GH that has 12 nodes: three nodes along the 0^{th} dimension and four nodes along the 1^{st} dimension.

In this work, we assume $k_i \geq 3$ because the one-to-many node disjoint paths (NDP) routing problem becomes trivial when $k_i < 3$. In the following, we describe some properties and concepts that are important for understanding the proposed algorithms.

Addressing: Any node x in a GH $Q^n_{k_{n-1},\ldots,k_1,k_0}$ can be addressed using an n-tuple $x = (x_{n-1} \ldots x_0) \in \mathbb{Z}_{k_{n-1}} \times \cdots \times \mathbb{Z}_{k_0}$. For example in Figure 2.1, the numbers inside the circles are the addresses. Each node x is addressed using a two-tuble $(x_1x_0) \in \mathbb{Z}_4 \times \mathbb{Z}_3$ where $\mathbb{Z}_4 = \{0, 1, 2, 3\}$ and $\mathbb{Z}_3 = \{0, 1, 2\}$.

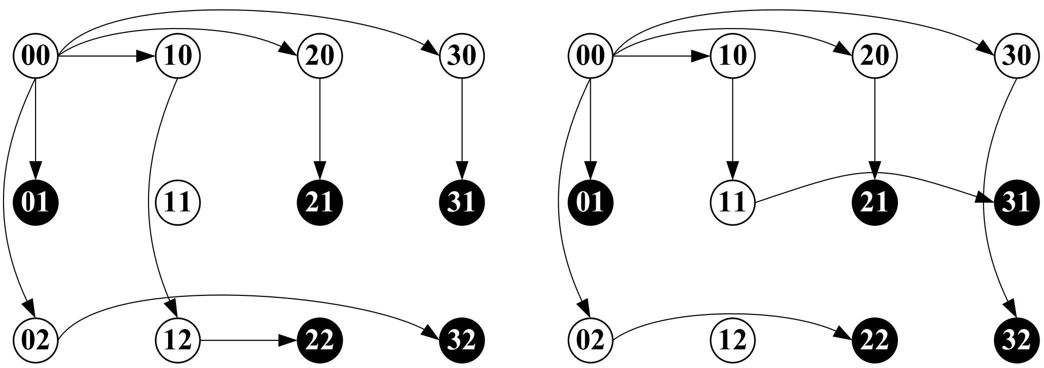
Connectivity: The Hamming distance $D_H(x, y)$ between node x and node y is the number of coordinates they differ along their addresses. In the GH, nodes x and y are neighbors (connected) if and only if the Hamming distance between them equals one (i.e. $D_H(x, y) = 1$). For example in Figure 2.1, nodes (02) and (32) are neighbors because $D_H(02, 32) = 1$, while nodes (02) and (11) are not neighbors because $D_H(02, 11) = 2$.

Diameter: The diameter is the largest possible distance between any two nodes in a network. In the GH, the diameter is equal to the number of dimensions n, because the addresses can differ at maximum in all n-coordinates. For example in Figure 2.1, the diameter equals two.

Degree: The node degree is the number of its neighbors. For any node x in $Q^n_{k_{n-1},\ldots,k_1,k_0}$, the node degree ℓ is equal to $\sum_{i=0}^{n-1}(k_i - 1)$ which is the number of x's neighbors. In the GH, all nodes have the same degree. Thus, the total number of links is $L = \frac{\ell.N}{2}$ where each link connects two neighbors. For example in Figure 2.1, $\ell = 5$ and $L = 30$.

Path: A path from node x to node y is denoted by $P(x, y) = \langle x, a1, a2, \ldots, a(|P(x, y)|-1), y\rangle$ where $|P(x, y)|$ is the length and each two consecutive nodes (e.g. x and $a1$) along the path are neighbors. The nodes $\langle a1, a2, \ldots, a(|P(x, y)| - 1)\rangle$ are called *internal nodes*. Sometimes, we write the path $P(x, y)$ as $x \to a1 \to a2 \to \cdots \to y$. The length of a shortest path from x to y is equal to the Hamming distance $D_H(x, y)$ between them. For example in Figure 2.1, one of the shortest paths between (00) and (32) is $00 \to 30 \to 32$, its length equals two which is equal to $D_H(00, 32)$. Another example of a longer path $P(00, 32)$ is $00 \to 01 \to 21 \to 22 \to 32$.

One-to-Many NDP: Given a source node s and a set of distinct destination nodes $T = \{t_1, t_2, \ldots, t_\ell\}$, where $s \notin T$ and ℓ is the node degree, a set of one-to-



(a) (b)

Figure 2.2: Different examples of NDP

many NDP connects s to each destination node t_i, $i \in \{1, 2, \ldots, \ell\}$, and satisfy the condition that the only common node among all paths is the source node s. Since the degree of each node in GH is ℓ, the maximum of destination nodes for which a set of NDP can be obtained from a given source node is also ℓ and this is the case in this work.

For a particular s and T, there are more than one possible set of NDP from s to T. One of these possible sets is denoted by $\mathbb{P}(s, T)$. For example consider the network in Figure 2.1, let the source node be $s = (00)$ and the set of destination nodes be $T = \{(01), (21), (22), (31), (32)\}$. Then, one possible set of NDP is $\mathbb{P}(s, T) = \{\langle 00, 01\rangle, \langle 00, 20, 21\rangle$, $\langle 00, 10, 12, 22\rangle, \langle 00, 30, 31\rangle, \langle 00, 02, 32\rangle\}$ (see Figure 2.2a). Another different possible set is $\mathbb{P}(s, T) = \{\langle 00, 01\rangle$, $\langle 00, 20, 21\rangle$, $\langle 00, 02, 22\rangle$, $\langle 00, 10, 11, 31\rangle$, $\langle 00, 30, 32\rangle\}$ (see Figure 2.2b).

Unreachable Destination Node: A destination node is called *unreachable destination node* when it is impossible to find a node disjoint path from the source node to that destination node. Constructing a set of NDP arbitrarily could lead to having unreachable destination nodes. For example consider the network in

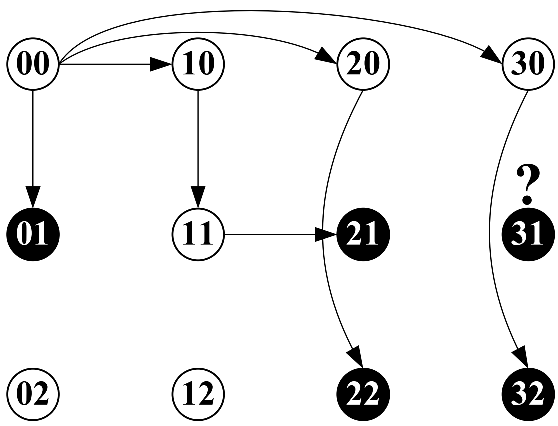


Figure 2.3: Example of unreachable destination node

Figure 2.1, let the source node be $s = (00)$ and the set of destination nodes be $T = \{(01), (21), (22), (31), (32)\}$. Suppose the following set of NDP have been constructed randomly (see Figure 2.3): $P(00, 01)$ as $00 \rightarrow 01$, $P(00, 21)$ as $00 \rightarrow 10 \rightarrow 11 \rightarrow 21$, $P(00, 22)$ as $00 \rightarrow 20 \rightarrow 22$, and $P(00, 32)$ as $00 \rightarrow 30 \rightarrow 32$. In this case, the destination node (31) is called unreachable destination node because we cannot add a path to this particular set without using one of the nodes more than once. Note that, the set $\mathbb{P}(s, T)$ reaches all destination nodes in T. Our proposed algorithms always return a solution $\mathbb{P}(s, T)$ to the one-to-many NDP routing problem without having unreachable destination nodes.

Unavailable Node: Any node in a GH $Q^n_{k_{n-1},\ldots,k_1,k_0}$ is called *unavailable node* if: 1) it is a destination node in T, or 2) an internal node along a path to one of the destination nodes. It is unavailable to be used during the process of constructing the NDP. Initially, all destination nodes in T are automatically unavailable nodes. As the construction process continues, each node that becomes an internal node also becomes unavailable node. Eventually, all nodes in the set $\mathbb{P}(s, T)$ are unavailable nodes.

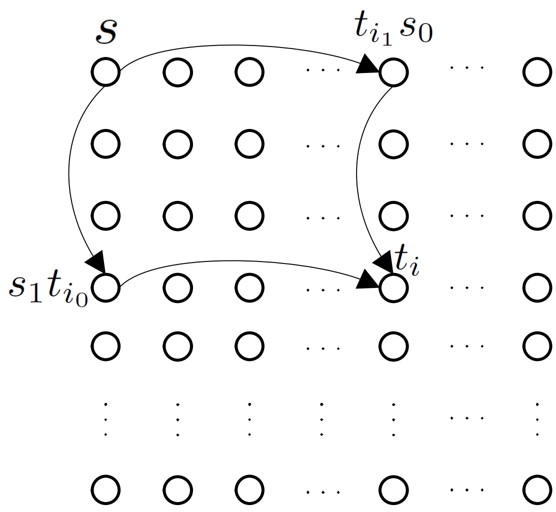


Figure 2.4: All shortest paths to t_i at distance two

In the following section, we introduce the proposed algorithm for two dimensional GH. In Section 2.3, we generalize this algorithm for any number of dimensions.

2.2 Routing in Two-Dimensional Generalized Hypercube

In this section, we propose an algorithm to solve the one-to-many node disjoint paths (NDP) routing problem in a two-dimensional GH denoted by $Q^2_{k_1,k_0}$. This problem is described as follows: given any source node $s = (s_1 s_0)$ and a set of distinct destination nodes $T = \{t_i = (t_{i_1} t_{i_0}) | 1 \leq i \leq \ell\}$ such that $s \notin T$ and $\ell = k_0 + k_1 - 2$, find a set of NDP $\mathbb{P}(s, T)$.

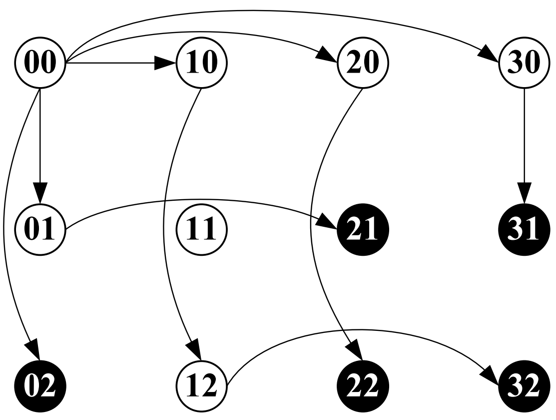


Figure 2.5: Solved example by Algorithm 1

2.2.1 Algorithm 1: Two-Dimensional GH

Before we describe the algorithm, notice that for any destination node $t_i = (t_{i_1} t_{i_0})$ at Hamming distance two from the source node s (i.e. $D_H(s, t_i) = 2$), there are only two shortest paths from s to t_i (see Figure 2.4): 1) $P(s, t_i)$ as $(s_1 s_0) \rightarrow (t_{i_1} s_0) \rightarrow (t_{i_1} t_{i_0})$, and 2) $P(s, t_i)$ as $(s_1 s_0) \rightarrow (s_1 t_{i_0}) \rightarrow (t_{i_1} t_{i_0})$. In this section, the node $(t_{i_1} s_0)$ is called *the column neighbor* of t_i (in the same column as t_i) while the node $(s_1 t_{i_0})$ is called *the row neighbor* of t_i (in the same row as t_i).

To explain the algorithm, we use the example in Figure 2.5 where the proposed algorithm is used to find a solution to this problem. In this example, the source node is $s = (00)$, the set of destination nodes is $T = \{(02), (21), (22), (31), (32)\}$, and the network is $Q^2_{4,3}$ as shown in Figure 2.1. The following steps describe the algorithm (see Algorithm 1):

Step 1 (Reach the source's neighbors)

In this step, the algorithm constructs a path from the source node to each destination node at Hamming distance one. For each destination node t_i such that $D_H(s, t_i) = 1$, the algorithm constructs the path $P(s, t_i)$ as $(s_1 s_0) \to (t_{i_1} t_{i_0})$. For example in Figure 2.5, the algorithm constructs the path $P(00, 02)$ as $00 \to 02$ in this step because $D_H(00, 02) = 1$.

Step 2 (Sort)

After reaching all destination nodes at Hamming distance one, the algorithm sorts the remaining destination nodes in ascending lexicographical order. For example in Figure 2.5, the ordered set of the remaining destination nodes (after reaching (02)) is $\langle 21, 22, 31, 32 \rangle$. This sorting is used to uniquely identify the last destination node within each column. Also, it is used to traverse the destination nodes in the network column by column.

Step 3 (Construct all paths)

Starting from the first destination node in the ordered set obtained in Step 2, the algorithm constructs a path from the source node s to each destination node in the ordered set $t_i = (t_{i_1} t_{i_0})$ according to the following cases:

Case 1: In this case: 1) t_i is not the last destination node in its column according to the sorting in Step 2, and 2) the row neighbor $(s_1 t_{i_0})$ or the column neighbor $(t_{i_1} s_0)$ is available.

Algorithm 1 One-to-Many NDP Routing in 2-Dimensional Generalized Hypercube

Input: $Q^2_{k_1,k_0}$, $s = (s_1 s_0)$, and $T = \{t_i = (t_{i_1} t_{i_0}) | 1 \le i \le \ell\}$ where $s \notin T$ and $\ell = k_0 + k_1 - 2$
Output: $\mathbb{P}(s, T)$

procedure $OneToMany\_2D(Q^2_{k_1,k_0}, s, T)$
  $Used = \{s, t_1, t_2, \ldots, t_\ell\}$; ▷ "*Used*" nodes
  $DP = \emptyset$; ▷ "*DP*" fully constructed NDP
  $Reached = \emptyset$; ▷ "*Reached*" dest. nodes
  for $1 \le i \le \ell$ **do** ▷ Step 1
    if $D_H(s, t_i) = 1$ **then**
      $DP = DP \cup \langle (s_1 s_0), (t_{i_1} t_{i_0}) \rangle$; $Reached = Reached \cup \{t_i\}$;
    end if
  end for
  Sort $RemT = T - Reached$ ▷ Step 2
  for $i \leftarrow 1, |RemT|$ **do** ▷ Step 3
    if t_i is *not* the last dest. node in its column **then**
      if $(s_1 t_{i_0}) \notin Used$ **then**
        $DP = DP \cup \langle (s_1 s_0), (s_1 t_{i_0}), (t_{i_1} t_{i_0}) \rangle$; $Used = Used \cup \{(s_1 t_{i_0})\}$;
        $Reached = Reached \cup \{t_i\}$;
      else if $(t_{i_1} s_0) \notin Used$ **then**
        $DP = DP \cup \langle (s_1 s_0), (t_{i_1} s_0), (t_{i_1} t_{i_0}) \rangle$; $Used = Used \cup \{(t_{i_1} s_0)\}$;
        $Reached = Reached \cup \{t_i\}$;
      else
        FindingPathOfLength_3;
      end if
    else ▷ t_i *is* the last destination node in its column
      if $(t_{i_1} s_0) \notin Used$ **then**
        $DP = DP \cup \langle (s_1 s_0), (t_{i_1} s_0), (t_{i_1} t_{i_0}) \rangle$; $Used = Used \cup \{(t_{i_1} s_0)\}$;
        $Reached = Reached \cup \{t_i\}$;
      else if $(s_1 t_{i_0}) \notin Used$ **then**
        $DP = DP \cup \langle (s_1 s_0), (s_1 t_{i_0}), (t_{i_1} t_{i_0}) \rangle$; $Used = Used \cup \{(s_1 t_{i_0})\}$;
        $Reached = Reached \cup \{t_i\}$;
      else
        FindingPathOfLength_3;
      end if
    end if
  end for
  $\mathbb{P}(s, T) = DP$;
  return $\mathbb{P}(s, T)$;
end procedure

Algorithm 1 Continued

procedure FINDINGPATHOFLENGTH_3
  $j = 0;\ \ isFound = false;$
  while $j \leq k_0 - 1$ and $isFound = false$ **do**
    $h = (t_{i_1}j); r = (s_1 j);$
    if $r \notin Used$ and $h \notin Used$ **then**
      $DP = DP \cup \langle (s_1 s_0), (s_1 j), (t_{i_1} j), (t_{i_1} t_{i_0}) \rangle;\ \ Used = Used \cup \{r, h\};$
      $Reached = Reached \cup \{t_i\}; isFound = true;$
    end if
    $j = j + 1;$
  end while
  if $isFound = false$ **then**
    $j = 0;$
    while $j \leq k_1 - 1$ and $isFound = false$ **do**
      $h = (j t_{i_0});\ \ \ r = (j s_0);$
      if $r \notin Used$ and $h \notin Used$ **then**
        $DP = DP \cup \langle (s_1 s_0), (j s_0), (j t_{i_0}), (t_{i_1} t_{i_0}) \rangle;\ Used = Used \cup \{r, h\};$
        $Reached = Reached \cup \{t_i\};\ \ isFound = true;$
      end if
      $j = j + 1;$
    end while
  end if
end procedure

To construct a path in this case, the algorithm first checks the availability of the row neighbor $(s_1 t_{i_0})$ of t_i, meaning that it has not been used so far in any path. If the row neighbor $(s_1 t_{i_0})$ is available, the algorithm constructs the path $P(s, t_i)$ as $(s_1 s_0) \to (s_1 t_{i_0}) \to (t_{i_1} t_{i_0})$. If the row neighbor $(s_1 t_{i_0})$ is not available, the algorithm checks the availability of the column neighbor $(t_{i_1} s_0)$ of t_i. If it is available, the algorithm constructs the path $P(s, t_i)$ as $(s_1 s_0) \to (t_{i_1} s_0) \to (t_{i_1} t_{i_0})$. If both (row and column) neighbors of t_i are not available, then go to Case 3.

One example of Case 1 is the destination node (21) in Figure 2.5. Since the row neighbor (01) is available at this point, the algorithm constructs the path $P(00, 21)$ as $00 \to 01 \to 21$. Another example is the destination node (31). Its row neighbor (01) is not available because it is an internal node in the path $P(00, 21)$. Since the row neighbor is not available, the algorithm checks the availability of the column neighbor (30) which is available at this point. So, the algorithm constructs the path $P(00, 31)$ as $00 \to 30 \to 31$.

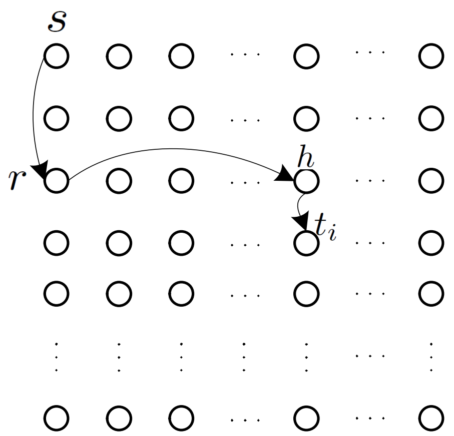
Case 2: In this case: 1) t_i is the last destination node in its column according to the sorting in Step 2, and 2) the row neighbor $(s_1 t_{i_0})$ or the column neighbor $(t_{i_1} s_0)$ is available.

Unlike the previous case, the algorithm first checks the availability of the column neighbor $(t_{i_1} s_0)$ of t_i, then the availability of the row neighbor $(s_1 t_{i_0})$ of t_i. If the column neighbor is available, the algorithm constructs the path $P(s, t_i)$ as $(s_1 s_0) \to (t_{i_1} s_0) \to (t_{i_1} t_{i_0})$. However, if the column neighbor is not available but the row neighbor $(s_1 t_{i_0})$ of t_i is available, the algorithm constructs the path $P(s, t_i)$ as $(s_1 s_0) \to (s_1 t_{i_0}) \to (t_{i_1} t_{i_0})$. If both (row and column) neighbors are not available, then go to Case 3.

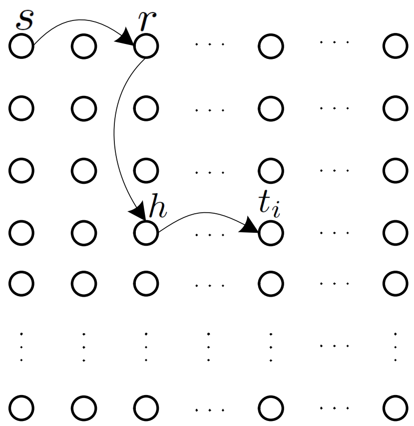
For example in Figure 2.5, to reach the destination node (22) which is the last node in its column, the algorithm first checks the availability of the column neighbor (20). The node (20) is available at this point. So, the algorithm constructs the path $P(00, 22)$ as $00 \to 20 \to 22$.

Case 3: In this case the row neighbor $(s_1 t_{i_0})$ and the column neighbor $(t_{i_1} s_0)$ are both not available, meaning that the shortest paths are not available.

The algorithm constructs a path of length three by finding an available neigh-



(a) Same column



(b) Same row

Figure 2.6: A path of length three (Case 3)

bor, h, of t_i such that h's neighbor which is in the same row or column as of s is also available. Let $h = (h_1 h_0)$ be any *available* neighbor of t_i, then h and t_i are either in the same column or in the same row (see Figure 2.6):

1. Same Column ($h = (t_{i_1} h_0)$): If node $r = (s_1 h_0)$ is available, the algorithm constructs the path $P(s, t_i)$ as $(s_1 s_0) \rightarrow (s_1 h_0) \rightarrow (t_{i_1} h_0) \rightarrow (t_{i_1} t_{i_0})$ (see Figure 2.6a).
2. Same Row ($h = (h_1 t_{i_0})$): If node $r = (h_1 s_0)$ is available, the algorithm constructs the path $P(s, t_i)$ as $(s_1 s_0) \rightarrow (h_1 s_0) \rightarrow (h_1 t_{i_0}) \rightarrow (t_{i_1} t_{i_0})$ (see Figure 2.6b).

As we prove it later, there is at least one available neighbor h such that its neighbor (either $(h_1 s_0)$ or $(s_1 h_0)$) is also available, meaning a path of length three exists.

For example in Figure 2.5, the shortest paths to the destination node (32) are not available because its column neighbor (30) is an internal node in the

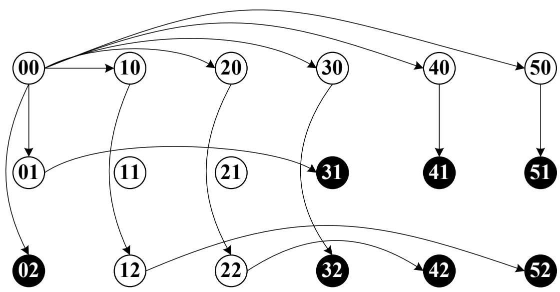


Figure 2.7: Solved example by Algorithm 1 ($Q^2_{6,3}$)

path $P(00, 31)$ and its row neighbor (02) is a destination node. Thus, the algorithm constructs a path of length three by finding an available neighbor h. The only available neighbor is $h = (12)$ which is in the same row as the destination node (32). Then, the algorithm checks the availability of node $(h_1 s_0) = (10)$ which is available. So, the algorithm constructs the path $P(00, 32)$ as $00 \rightarrow 10 \rightarrow 12 \rightarrow 32$.

Example 2.2.1 provides a complete example of Algorithm 1.

Example 2.2.1. *Consider the GH* $Q^2_{6,3}$. *The node degree in this GH is* $\ell = 7$. *Let the source node be* $s = (00)$ *and the set of the seven destination nodes be* $T = \{(52), (51), (41), (32), (02), (42), (31)\}$. *Algorithm 1 solves this example as follows (see Figure 2.7):*

Step 1: *Construct a path to each destination node at Hamming distance one:*

- $P(00, 02)$ *is* $00 \rightarrow 02$

Step 2: *Sort the remaining destination nodes:*

- $\langle 31, 32, 41, 42, 51, 52 \rangle$

Step 3: *Construct a path to each one of the remaining destination nodes:*

- $P(00,31)$ *is* $00 \to 01 \to 31$ *(Case 1: check* (01) *then* (30)*)*
- $P(00,32)$ *is* $00 \to 30 \to 32$ *(Case 2: check* (30) *then* (02)*)*
- $P(00,41)$ *is* $00 \to 40 \to 41$ *(Case 1: check* (01) *then* (40)*,* (01) *has been used by* $P(00,31)$*)*
- $P(00,42)$ *is* $00 \to 20 \to 22 \to 42$ *(Case 3:* (22) *and its neighbor* (20) *are both available)*
- $P(00,51)$ *is* $00 \to 50 \to 51$ *(Case 1: check* (01) *then* (50)*,* (01) *has been used by* $P(00,31)$*)*
- $P(00,52)$ *is* $00 \to 10 \to 12 \to 52$ *(Case 3:* (12) *and its neighbor* (10) *are both available)*

All destination nodes have been reached using a set of NDP.

■

2.2.2 Correctness of Algorithm 1

We prove the correctness of Algorithm 1 by providing the following theorem and its proof.

Theorem 2.2.1. *In a GH* $Q^2_{k_1,k_0}$*, given any source node* $s = (s_1 s_0)$ *and a set of distinct destination nodes* $T = \{t_i = (t_{i_1} t_{i_0}) | 1 \leq i \leq \ell\}$ *such that* $s \notin T$ *and* $\ell = k_0 + k_1 - 2$*, Algorithm 1 always finds a set of NDP* $\mathbb{P}(s,T)$ *with path lengths at most three.*

Proof. Since the GH is a symmetric network, without loss of generality, assume that the source node s is (00). Let $t_i = (t_{i_1}t_{i_0}) \in T$ be the current destination node in Algorithm 1. Then, there are three distinct cases:

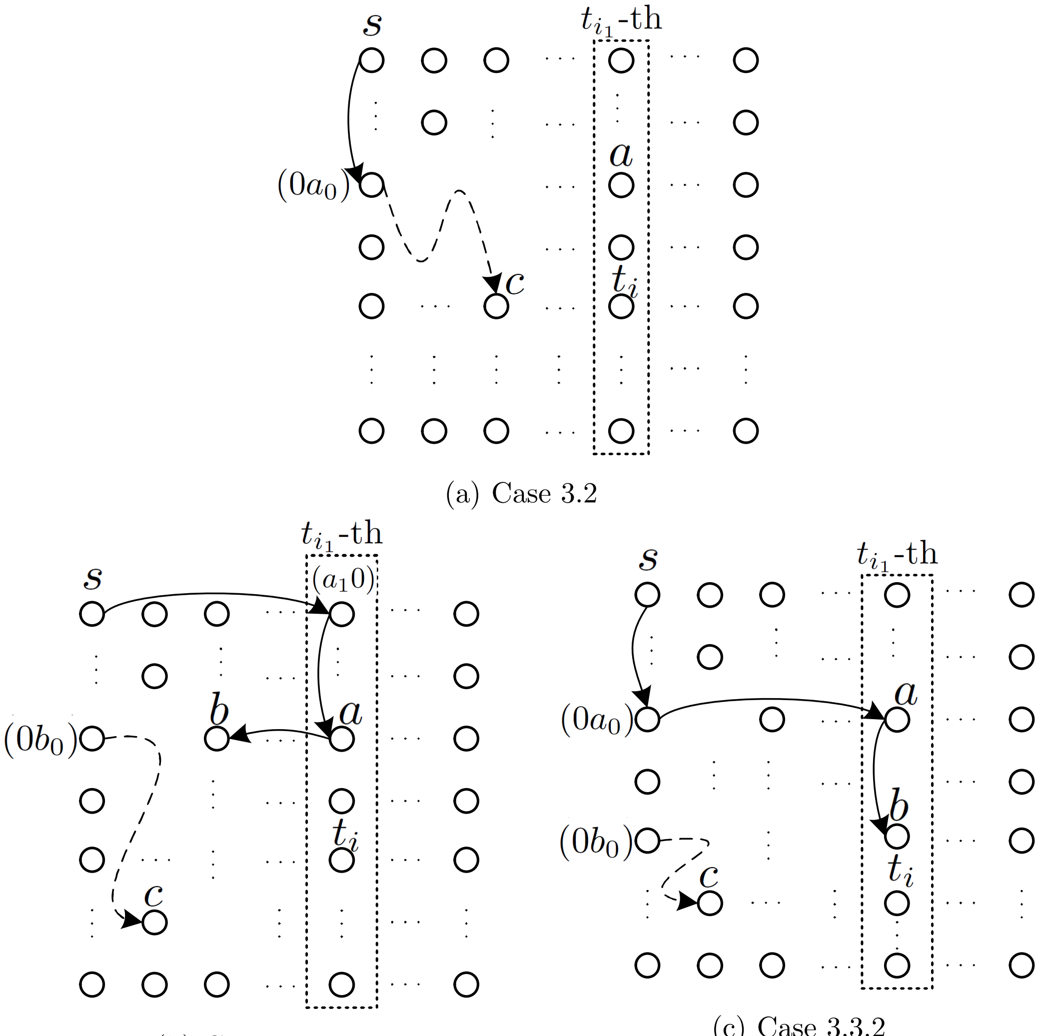
Case 1: Suppose t_i and s are neighbors. In this case (by Step 1 of Algorithm 1), the path $P(s,t_i)$ is $(00) \rightarrow (t_{i_1}t_{i_0})$ and its length is equal to one.

Case 2: Suppose t_i and s are not neighbors and the column neighbor $(t_{i_1}0)$, the row neighbor $(0t_{i_0})$, or both are available. In this case (by either Case 1 or Case 2 of Step 3 of Algorithm 1), the path $P(s,t_i)$ is $(00) \rightarrow (0t_{i_0}) \rightarrow (t_{i_1}t_{i_0})$ or $(00) \rightarrow (t_{i_1}0) \rightarrow (t_{i_1}t_{i_0})$ (see Figure 2.4) and its length is equal to two.

Case 3: Suppose t_i and s are not neighbors and the column neighbor $(t_{i_1}0)$ and the row neighbor $(0t_{i_0})$ are both *not* available. In this case, we need to prove that there exists a path of length three from s to t_i (Case 3 of Step 3 of Algorithm 1). We prove that by contradiction.

Assume a node disjoint path $P(s,t_i)$ of length three does not exist. Then, we prove that there exist more than $k_0 + k_1 - 2$ destination nodes, which is a contradiction because it is assumed that there are exactly $k_0 + k_1 - 2$ destination nodes.

Let us first consider the column that has the destination node t_i (t_{i_1}-th). After excluding t_i, the number of nodes in this column is equal to $k_0 - 1$. In the following we show that there exists a one-to-one correspondence between all nodes other than t_i in the t_{i_1}-th column and distinct destination nodes. Let $a = (t_{i_1}a_0)$ be any node in the t_{i_1}-th column other than t_i (i.e. $a_0 \neq t_{i_0}$). We have the following cases:



(a) Case 3.2

(b) Case 3.3.1

(c) Case 3.3.2

Figure 2.8: Proof of Case 3

Case 3.1: Suppose node a is a destination node. In this case, the corresponding destination node is node a itself.

Case 3.2: Suppose node a is not a destination node but it has not been used (an available node). Then the node $(0a_0)$ must be used to reach a destination node $c = (c_1c_0)$ such that $0 < c_1 < a_1$ (see Figure 2.8a). In this case, the corresponding destination node for a is node c.

Case 3.3: Suppose node a is not a destination node but it *has* been used (an unavailable node). Then there are two cases:

Case 3.3.1: Suppose node a has been used by a path $P(s, b)$ of length three from s to a destination node $b = (b_1 a_0)$ as $(00) \to (a_1 0) \to (a_1 a_0) \to (b_1 a_0)$ (see Figure 2.8b). In this case, the corresponding destination node for node a is node b. Note that the node $(0 b_0)$ must be used to reach a destination node $c = (c_1 c_0)$ such that $0 < c_1 < b_1$. So, the corresponding destination node for node $(a_1 0)$ is node c.

Case 3.3.2: Suppose node a has been used by a path $P(s, b)$ of length three from s to a destination node $b = (a_1 b_0)$ as $(00) \to (0 a_0) \to (a_1 a_0) \to (a_1 b_0)$ (see Figure 2.8c). In this case, the corresponding destination node for node b is itself. Note that the node $(0 b_0)$ must be used to reach a destination node $c = (c_1 c_0)$ such that $0 < c_1 < b_1$. So, the corresponding destination node for node a is node c.

From the above argument, we have found a one-to-one correspondence between all nodes other than t_i in the t_{i_1}-th column and distinct destination nodes. It follows that we have counted $k_0 - 1$ distinct destination nodes so far. These $k_0 - 1$ destination nodes are either in the t_{i_1}-th column or in the path from s to that node goes through $(t_{i_1} 0)$ or through a node other than $(0 t_{i_0})$ in the first column.

Using a similar argument, we can prove that there is a one-to-one correspondence between the nodes in the t_{i_0}-th row (other than t_i) and a set of $k_1 - 1$ distinct destination nodes. These $k_1 - 1$ destination nodes are either in the t_{i_0}-th row or in the path from s to that node goes through $(0t_{i_0})$ or through a node in the first row other than $(t_{i_1}0)$. Thus, these destination nodes are different from the $k_0 - 1$ destination nodes obtained in the above argument. As a result, including the destination node t_i, we get a total of $(k_0 - 1) + (k_1 - 1) + 1 = k_0 + k_1 - 1$ destination nodes. This gives a contradiction because it is assumed that there are exactly $k_0 + k_1 - 2$ destination nodes. So, there exists a path $P(s, t_i)$ of length three .

This proves that Algorithm 1 is correct. □

Corollary 2.2.1. *For any node disjoint path $P(s, t_i)$ in $\mathbb{P}(s, T)$ generated by Algorithm 1, the upper and lower bounds of the path length are given in Inequation* (2.1).

$$D_H(s, t_i) \leq |P(s, t_i)| \leq 3 \tag{2.1}$$

The time complexity of Algorithm 1 is same as the time complexity of Algorithm 2 with $n = 2$ (the time complexity analysis for Algorithm 2 is given in Section 2.3.3). Thus, the time complexity of Algorithm 1 is $O({k_{max}}^2)$ where $k_{max} = \max\{k_0, k_1\}$.

In the following section, we generalize Algorithm 1 to solve the same problem for any number of dimensions (i.e $n \geq 2$). In Section 2.4, we show the simulation results for both algorithms.

2.3 Routing in n-Dimensional Generalized Hypercube

In this section, we propose Algorithm 2 which solves the one-to-many node disjoint paths (NDP) routing problem in n-dimensional GH denoted by $Q^n_{k_{n-1},\ldots,k_1,k_0}$. This

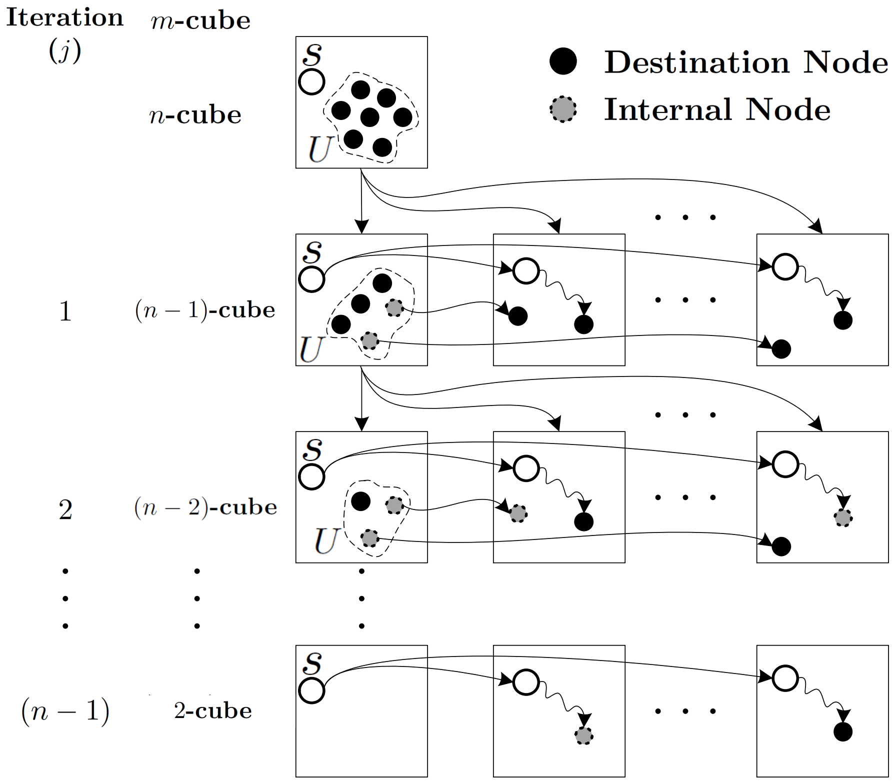


Figure 2.9: The basic idea of Algorithm 2

problem is described as follows: given any source node $s = (s_{n-1} \dots s_1 s_0)$ and a set of distinct destination nodes $T = \{t_i = (t_{i_{n-1}} \dots t_{i_1} t_{i_0}) | 1 \leq i \leq \ell\}$ such that $s \notin T$ and $\ell = \sum_{i=0}^{n-1}(k_i - 1)$, find a set of NDP $\mathbb{P}(s, T)$ from s to each destination node in T.

2.3.1 Algorithm 2: n-Dimensional GH

Before explaining the detailed steps of Algorithm 2, we first give an overview. Algorithm 2 is an iterative algorithm (see Figure 2.9). Let $j = 1, 2, \dots, (n-1)$ be the iteration counter. In the first iteration ($j = 1$), the algorithm partitions the n-dimensional GH to a number of mutually disjoint $(n-1)$-dimensional subcubes, satisfying some properties to be discussed soon. In the second iteration ($j = 2$), the algorithm partitions the $(n-1)$-dimensional subcube that has the source node to a number of mutually disjoint $(n-2)$-dimensional subcubes, again satisfying some properties to be discussed soon. Let $m = n - (j-1)$ be the dimension of the source's subcubes (before partitioning) in the current iteration j. So in each iteration, the algorithm partitions the m-dimensional cube that has the source node to a number of mutually disjoint $(m-1)$-dimensional cubes.

The partitioning process (as explained later) depends on the number of *unavailable nodes* (as defined in Section 2.1) in the subcube that has the source node. Let $U = \{u_i = (u_{i_{m-1}} \dots u_{i_0}) | 1 \leq i \leq \sum_{j=0}^{m-1}(k_j - 1)\}$ be a set of unavailable nodes in the m-dimensional cube that has the source node. As we have defined earlier, an unavailable node u_i is either a destination node (i.e. $u_i \in T$) or an internal node. Initially when ($j = 1$), U contains only all destination nodes in T (i.e. $U = T$). After that, U has different unavailable nodes in each iteration as the algorithm performs the following operations on the resultant subcubes (after partitioning) during each iteration (see Figure 2.9 and note that the source node exists exactly in one of the resultant subcubes after each partitioning):

1. For each subcube other than the source's subcube, Algorithm 2 designates one unavailable node; and the path to that node goes from the source node to the source's immediate neighbor in that subcube and then to that unavailable node.

2. For all other unavailable nodes (other than those considered in the above step and not existed in the source's subcube), the algorithm *maps* them (as explained later) to distinct nodes in the subcube that has the source node.

000	020	100	120	200	220
001	021	101	121	201	221
002	022	102	122	202	222
010	030	110	130	210	230
011	031	111	131	211	231
012	032	112	132	212	232

Figure 2.10: All nodes of $Q^3_{3,4,3}$ (the source node is (000) and the destination nodes are in black)

> The mapping process includes adding these distinct nodes to the set of NDP. So, they become internal nodes and therefore unavailable nodes. Since they are in the subcube that has the source node, the algorithm adds them to U. Thus, they will be considered during the next partitioning. Note that, U also contains any unavailable node that happens to be in the source's subcube as a result of the partitioning.

For example in Figure 2.9, as a result of the partitioning and mapping during the first iteration, the second iteration starts with U that has three destination nodes and two internal nodes. Similarly, as a result of the partitioning and mapping during the second iteration, the third iteration starts with U that has one destination node and two internal nodes. Algorithm 2 constructs each path starting from the destination node and keeps adding internal node(s) in each iteration (not necessary in each and every iteration) until the path is fully constructed.

To explain the detailed steps of Algorithm 2, we use the example in Figure 2.10. In this example, the source node is $s = (000)$, the set of destination nodes is $T = \{(001), (021), (031), (002), (022), (032), (102)\}$, and the network is $Q^3_{3,4,3}$. The following steps describe the algorithm in details (see Algorithm 2):

$q_0^{2,0}$

000	120
010	130
020	200
030	210
100	220
110	230

$q_0^{2,1}$

001	121
011	131
021	201
031	211
101	221
111	231

$q_0^{2,2}$

002	122
012	132
022	202
032	212
102	222
112	232

(a) $\check{e} = 0, \ell_0 = 5$

$q_1^{2,0}$

000	102
001	200
002	201
100	202
101	

$q_1^{2,1}$

010	112
011	210
012	211
110	212
111	

$q_1^{2,2}$

020	122
021	220
022	221
120	222
121	

$q_1^{2,3}$

030	132
031	230
032	231
130	232
131	

(b) $\check{e} = 1, \ell_1 = 4$

$q_2^{2,0}$

000	020
001	021
002	022
010	030
011	031
012	032

$q_2^{2,1}$

100	120
101	121
102	122
110	130
111	131
112	132

$q_2^{2,2}$

200	220
201	221
202	222
210	230
211	231
212	232

(c) $\check{e} = 2, \ell_2 = 5$

Figure 2.11: All ways to partition $Q^3_{3,4,3}$

Step 1 (Reach the source's neighbors)

For each destination node t_i such that $D_H(s, t_i) = 1$, the path $P(s, t_i)$ as $(s_{n-1} \dots s_0)$ $\rightarrow$ $(t_{i_{n-1}} \dots t_{i_0})$ is constructed in this step. For example in Figure 2.10, the algorithm constructs in this step the paths $P(000, 001)$ as $000 \rightarrow 001$ and $P(000, 002)$ as $000 \rightarrow 002$ because $D_H(000, 001) = D_H(000, 002) = 1$. This step is performed only once.

Step 2 (Partition)

In this step, the algorithm partitions the m-dimensional cube , denoted by $Q^m_{k_{m-1},\ldots,k_1,k_0}$, that has the source node using dimension $\check{e} \in \{0, 1, 2, \ldots, m-1\}$ to $k_{\check{e}}$ mutually disjoint $(m-1)$-dimensional subcubes: $q_{\check{e}}^{m-1,0}, q_{\check{e}}^{m-1,1}, \ldots, q_{\check{e}}^{m-1,k_{\check{e}}-1}$. The subcube $q_{\check{e}}^{m-1,x}$ is obtained by fixing the $\check{e}$-th coordinate to x where $x \in \{0, 1, \ldots, (k_{\check{e}} - 1)\}$. The node degree $\ell_{\check{e}}$ in these subcubes is equal to $\ell_{\check{e}} = \sum_{i=0,i\neq\check{e}}^{m-1}(k_i - 1)$.

Clearly, there are m different ways to partition the m-dimensional cube. For example in Figure 2.10, during the first iteration ($j = 1, m = n = 3$), $Q^3_{3,4,3}$ can be partitioned in three different ways (see Figure 2.11):

1. $\check{e} = 0$ (Figure 2.11a): $q_0^{2,0}, q_0^{2,1}, q_0^{2,2}$. The node degree in these subcubes is $\ell_0 = 3 + 2 = 5$.

2. $\check{e} = 1$ (Figure 2.11b): $q_1^{2,0}, q_1^{2,1}, q_1^{2,2}, q_1^{2,3}$. The node degree in these subcubes is $\ell_1 = 2 + 2 = 4$.

3. $\check{e} = 2$ (Figure 2.11c): $q_2^{2,0}, q_2^{2,1}, q_2^{2,2}$. The node degree in these subcubes is $\ell_2 = 2 + 3 = 5$.

Algorithm 2 One-to-Many NDP Routing in n-Dimensional Generalized Hypercube

Input: $Q^n_{k_{n-1},\ldots,k_1,k_0}$, $s = (s_{n-1}\ldots s_1 s_0)$, and $T = \{t_i = (t_{i_{n-1}}\ldots t_{i_1} t_{i_0}) | 1 \le i \le \ell\}$ where $s \notin T$ and $\ell = \sum_{i=0}^{n-1}(k_i - 1)$

Output: $\mathbb{P}(s, T)$

procedure $OneToMany\_nD(Q^n_{k_{n-1},\ldots,k_1,k_0}, s, T)$
  $Used = \{s, t_1, t_2, \ldots, t_\ell\}$; $DP = \emptyset$; $Reached = \emptyset$;
  for $1 \le i \le \ell$ **do** ▷ Step 1: Reach the Source's Neighbors
    if $D_H(s, t_i) = 1$ **then**
      $P(s, t_i) = \big\langle (s_{n-1}\ldots s_0), (t_{i_{n-1}}\ldots t_{i_0}) \big\rangle$; $DP = DP \cup P(s, t_i)$; $Reached = Reached \cup \{t_i\}$;
    end if
  end for
  $j = 1$; $m = n$; $U = T$; ▷ U is the set of unavailable nodes in the source's subcube
  while $|Reached| \neq |T|$ **do**
    $e = \arg\max_{0 \le \check{e} \le m-1} \left\{ v_{\check{e}}^{(m-1, s_{\check{e}})} | v_{\check{e}}^{(m-1,x)} \le \ell_{\check{e}} = \sum_{i=0, i\neq \check{e}}^{m-1}(k_i - 1) \quad \forall \quad x \in \{0, 1, \ldots, (k_{\check{e}} - 1)\} \right\}$; ▷ Step 2: Partition
    Partition $Q^m_{k_{m-1},\ldots,k_1,k_0}$ to $q_e^{m-1,0}, q_e^{m-1,1}, \ldots, q_e^{m-1,k_e-1}$; ▷ $Q^m_{k_{m-1},\ldots,k_1,k_0}$ is the m-cube that has the source
    $U = U \cap q_e^{m-1,s_e}$; ▷ Reset U to have all unavailable nodes in the new source's subcube
    $v_e^{(m-1,s_e)} = |U|$;
    Sort the subcubes other than q_e^{m-1,s_e} in the following order: $q_e^{m-1,0}, q_e^{m-1,1}, \ldots, q_e^{m-1,k_e-1}$; ▷ Step 3: Sort
    Sort the unavailable nodes within each subcube other than q_e^{m-1,s_e} in ascending lexicographical order ;
    STEP_4 ; ▷ Step 4: Construct or Map
    $j = j + 1$; $m = m - 1$;
  end while
  $\mathbb{P}(s, T) = DP$;
  return $\mathbb{P}(s, T)$;
end procedure

Algorithm 2 Continued

```
procedure STEP_4
    for each subcubes $q_e^{m-1,0}, q_e^{m-1,1}, \ldots, q_e^{m-1,k_e-1}$ other than $q_e^{m-1,s_e}$ do
        for each unavailable node $u_i$ in this subcube do
            if $u_i$ is not the last unavailable node in $q_e^{m-1,u_{ie}}$ then
                if $u_i^{(e,s_e)} \notin Used$ and $v_e^{(m-1,s_e)} < \ell_e$ then
                    CASES_1_6 ;                                        ▷ Case 1
                else if $s^{(e,u_{ie})} \notin Used$ then
                    CASES_2_5 ;                                        ▷ Case 2
                else
                    FIND_H
                    if $h$ and $h^{(e,s_e)}$ exist and $v_e^{(m-1,s_e)} < \ell_e$ then
                        CASES_3_7 ;                                    ▷ Case 3
                    else
                        CASES_4_8 ;                                    ▷ Case 4
                    end if
                end if
            else if $s^{(e,u_{ie})} \notin Used$ then
                CASES_2_5 ;                                            ▷ Case 5
            else if $u_i^{(e,s_e)} \notin Used$ and and $v_e^{(m-1,s_e)} < \ell_e$ then
                CASES_1_6 ;                                            ▷ Case 6
            else
                FIND_H
                if $h$ and $h^{(e,s_e)}$ exist and $v_e^{(m-1,s_e)} < \ell_e$ then
                    CASES_3_7 ;                                        ▷ Case 7
                else
                    CASES_4_8 ;                                        ▷ Case 8
                end if
            end if
        end for
    end for
end procedure
```

Algorithm 2 Continued

procedure FIND_H
  $G = \{g_i | g_i \in q_e^{m-1,u_{i_e}} \quad and \quad D_H(g_i, u_i) = 1\}$;
  Sort G in ascending order according to $D_H(s, g_i)$;
  for each $g_i \in G$ **do**
    $g_i^{(e,s_e)} = x$ s.t. $D_H(g_i, x) = 1 \quad and \quad x \in q_e^{m-1,s_e}$;
    if $g_i \notin Used$ and $g_i^{(e,s_e)} \notin Used$ **then**
      $h = g_i; \quad h^{(e,s_e)} = g_i^{(e,s_e)}$;
    end if
  end for
end procedure

Algorithm 2 Continued

procedure CASES_1_6
  $P(s, t_i) = \big\langle (s_{m-1} \ldots s_0), \ldots, (u_{i_{m-1}} \ldots u_{i_{e+1}} s_e u_{i_{e-1}} \ldots u_{i_0}), (u_{i_{m-1}} \ldots u_{i_0}), \ldots, (t_{i_{m-1}} \ldots t_{i_0}) \big\rangle$;
  $Used = Used \cup (u_{i_{m-1}} \ldots u_{i_{e+1}} s_e u_{i_{e-1}} \ldots u_{i_0})$;
  $U = U \cup (u_{i_{m-1}} \ldots u_{i_{e+1}} s_e u_{i_{e-1}} \ldots u_{i_0})$;
  $v_e^{(m-1,s_e)} = v_e^{(m-1,s_e)} + 1$;
  if $u_i^{(e,s_e)}$ and s are neighbors **then**
    $DP = DP \cup P(s, t_i); Reached = Reached \cup \{t_i\}$;
  end if
end procedure

Algorithm 2 Continued

procedure CASES_2_5
  $P(s, t_i) = \big\langle (s_{m-1} \ldots s_0), (s_{m-1} \ldots s_{e+1} u_{i_e} s_{e-1} \ldots s_0), \ldots, (u_{i_{m-1}} \ldots u_{i_0}), \ldots, (t_{i_{m-1}} \ldots t_{i_0}) \big\rangle$;
  $DP = DP \cup P(s, t_i); Reached = Reached \cup \{t_i\}$;
  $Used = Used \ \cup$ all internal nodes in $P(s, u_i)$;
end procedure

Algorithm 2 Continued

procedure CASES_3_7
    $P(s,t_i) = \big\langle(s_{m-1}\dots s_0),\dots,\ (u_{i_{m-1}}\dots u_{i_{e+1}}s_e u_{i_{e-1}}\dots h_j\dots u_{i_0}),$ $(u_{i_{m-1}}\dots u_{i_{e+1}}u_{i_e}u_{i_{e-1}}\dots h_j\dots u_{i_0}),(u_{i_{m-1}}\dots u_{i_0}),\dots,(t_{i_{m-1}}\dots t_{i_0})\big\rangle;$
    $Used = Used \cup \{(u_{i_{m-1}}\dots u_{i_{e+1}}s_e u_{i_{e-1}}\dots h_j\dots u_{i_0}),$ $(u_{i_{m-1}}\dots u_{i_{e+1}}u_{i_e}u_{i_{e-1}}\dots h_j\dots u_{i_0})\};$
    $U = U \cup (u_{i_{m-1}}\dots u_{i_{e+1}}s_e u_{i_{e-1}}\dots h_j\dots u_{i_0});$
    $v_e^{(m-1,s_e)} = v_e^{(m-1,s_e)} + 1;$
    if $h^{(e,s_e)}$ and s are neighbors **then**
        $DP = DP \cup P(s,t_i); Reached = Reached \cup \{t_i\};$
    end if
end procedure

Algorithm 2 Continued

procedure CASES_4_8
    $P(s,t_i) = \big\langle(s_{m-1}\dots s_0),\ (s_{i_{m-1}}\dots s_{i_{e+1}}p s_{i_{e-1}}\dots s_{i_0}),\dots,$ $(u_{i_{m-1}}\dots u_{i_{e+1}}p u_{i_{e-1}}\dots u_{i_0}),(u_{i_{m-1}}\dots u_{i_0}),\dots,(t_{i_{m-1}}\dots t_{i_0})\big\rangle;$
    $DP = DP \cup P(s,t_i); Reached = Reached \cup \{t_i\};$
    $Used = Used \cup$ all internal nodes in $P(s,t_i)$;
end procedure

Choosing the partitioning dimension $\check{e}$ is crucial to avoid the unreachable destination node problem. Note that the unavailable nodes are distributed differently depending on the choice of the partitioning dimension $\check{e}$. Let $v_{\check{e}}^{(m-1,x)}$ be the number of unavailable nodes in the subcube $q_{\check{e}}^{m-1,x}$ where $x \in \{0,1,\dots,(k_{\check{e}}-1)\}$. So, $v_{\check{e}}^{(m-1,s_{\check{e}})}$ is equal to the number of unavailable nodes in the subcube that has the source node such that the m-dimensional cube was partitioned using the partitioning dimension $\check{e}$. For example in Figure 2.11b, since $s = (s_2s_1s_0) = (000)$, the number of unavailable nodes $v_1^{(m-1,s_1)} = v_1^{(2,0)}$ in the subcube that has the source node is equal to three. Similarly, $v_1^{(2,1)} = 0$, $v_1^{(2,2)} = 2$, and $v_1^{(2,3)} = 2$. Let

$$e \;=\; \underset{0\le\check{e}\le m-1}{\arg\max}\left\{ v_{\check{e}}^{(m-1,s_{\check{e}})} | v_{\check{e}}^{(m-1,x)} \;\le\; \ell_{\check{e}} \quad \forall \quad x \;\in\; \{0,1,\ldots,(k_{\check{e}}\;-\;1)\}\right\} \tag{2.2}$$

Algorithm 2 uses Equation (2.2) to choose the partitioning dimension e. This equation returns the partitioning dimension e, such that the subcube containing the source node has the highest but less than or equal ℓ_e number of unavailable nodes. Furthermore, all other subcubes contain at most ℓ_e unavailable nodes. There exists at least one partitioning dimension e satisfying Equation (2.2) (to be proved in Section 2.3.2).

For example in Figure 2.11, the 2$^{\text{nd}}$ dimension ($\check{e} = 2$, Figure 2.11c) will not be chosen because the number of unavailable nodes $v_2^{(2,0)} = 6$ in the subcube $q_2^{2,0}$ is more than the node degree $\ell_2 = 5$. The algorithm will choose the 1$^{\text{st}}$ dimension (i.e. $e = 1$) because the number of unavailable nodes $v_1^{(2,0)} = 3 \le \ell_1 = 4$ in $q_1^{2,0}$ is more than the number of unavailable nodes $v_0^{(2,0)} = 0 \le \ell_0 = 5$ in $q_0^{2,0}$ and all subcubes $q_1^{2,0}$, $q_1^{2,1}$, $q_1^{2,2}$, and $q_1^{2,3}$ have less than or equal $\ell_1 = 4$ unavailable nodes. Note that $q_2^{2,0}$ can contain at most $\ell_1 = 4$ unavailable nodes.

Step 3 (Sort)

After the partitioning process, the algorithm creates an ordered set of unavailable nodes that are not in the source's subcube q_e^{m-1,s_e} by first sorting the subcubes other than q_e^{m-1,s_e} in the following order: $q_e^{m-1,0}, q_e^{m-1,1}, \ldots, q_e^{m-1,k_e-1}$. Then, within each subcube in this order, the algorithm sorts the unavailable nodes in ascending lexicographical order. For example in Figure 2.11b, the resultant ordered set is $\langle 021, 022, 031, 032\rangle$. This sorting is used to uniquely identify the last destination node in each subcube.

Step 4 (Construct or map)

Starting from the first unavailable node u_i in the ordered set obtained in Step 3, the algorithm either constructs the path $P(s, t_i)$ completely as $s \to \cdots \to u_i \to \cdots \to t_i$ or adds one or two internal node(s) (mapping) to the portion $s \to \cdots \to u_i$ of this path, according to the following cases (see Figure 2.12):

Case 1: Suppose the following: 1) u_i is not the last unavailable node in its subcube $q_e^{m-1,u_{i_e}}$ (according to the ordered set obtained in Step 3), 2) the current number $v_e^{(m-1,s_e)}$ of unavailable nodes in the source's subcube is less than the node degree in the source's subcube (i.e. $v_e^{(m-1,s_e)} < \ell_e$), and 3) the neighbor of u_i in the source's subcube is available. Note that the address of this neighbor is $(u_{i_{m-1}} \dots u_{i_{e+1}} s_e u_{i_{e-1}} \dots u_{i_0})$, which we represent as $u_i^{(e,s_e)}$ (i.e. replace the e-th coordinate of u_i's address by s_e).

In Case 1, the algorithm maps u_i by adding its neighbor $u_i^{(e,s_e)}$ to the path $P(s, t_i)$ as $(s_{m-1} \dots s_0) \to \cdots \to (u_{i_{m-1}} \dots u_{i_{e+1}} s_e u_{i_{e-1}} \dots u_{i_0}) \to$ $(u_{i_{m-1}} \dots u_{i_0}) \to \cdots \to (t_{i_{m-1}} \dots t_{i_0})$. In the next iteration, $u_i^{(e,s_e)}$ is one of the unavailable nodes in U. If s and $u_i^{(e,s_e)}$ are neighbors, the path $P(s, t_i)$ has been connected as $s \to u_i^{(e,s_e)} \to u_i \to \cdots \to t_i$. This action increases $v_e^{(m-1,s_e)}$ by one because $u_i^{(e,s_e)}$ becomes unavailable node.

Case 2: Suppose the following: 1) u_i is not the last unavailable node in $q_e^{m-1,u_{i_e}}$, 2) $u_i^{(e,s_e)}$ is not available or $v_e^{(m-1,s_e)} = \ell_e$, and 3) the neighbor of s in the subcube that has u_i is available. This neighbor is $s^{(e,u_{i_e})} =$ $(s_{m-1} \dots s_{e+1} u_{i_e} s_{e-1} \dots s_0)$.

In Case 2, the algorithm constructs the path $P(s, t_i)$ completely as $(s_{m-1} \dots$

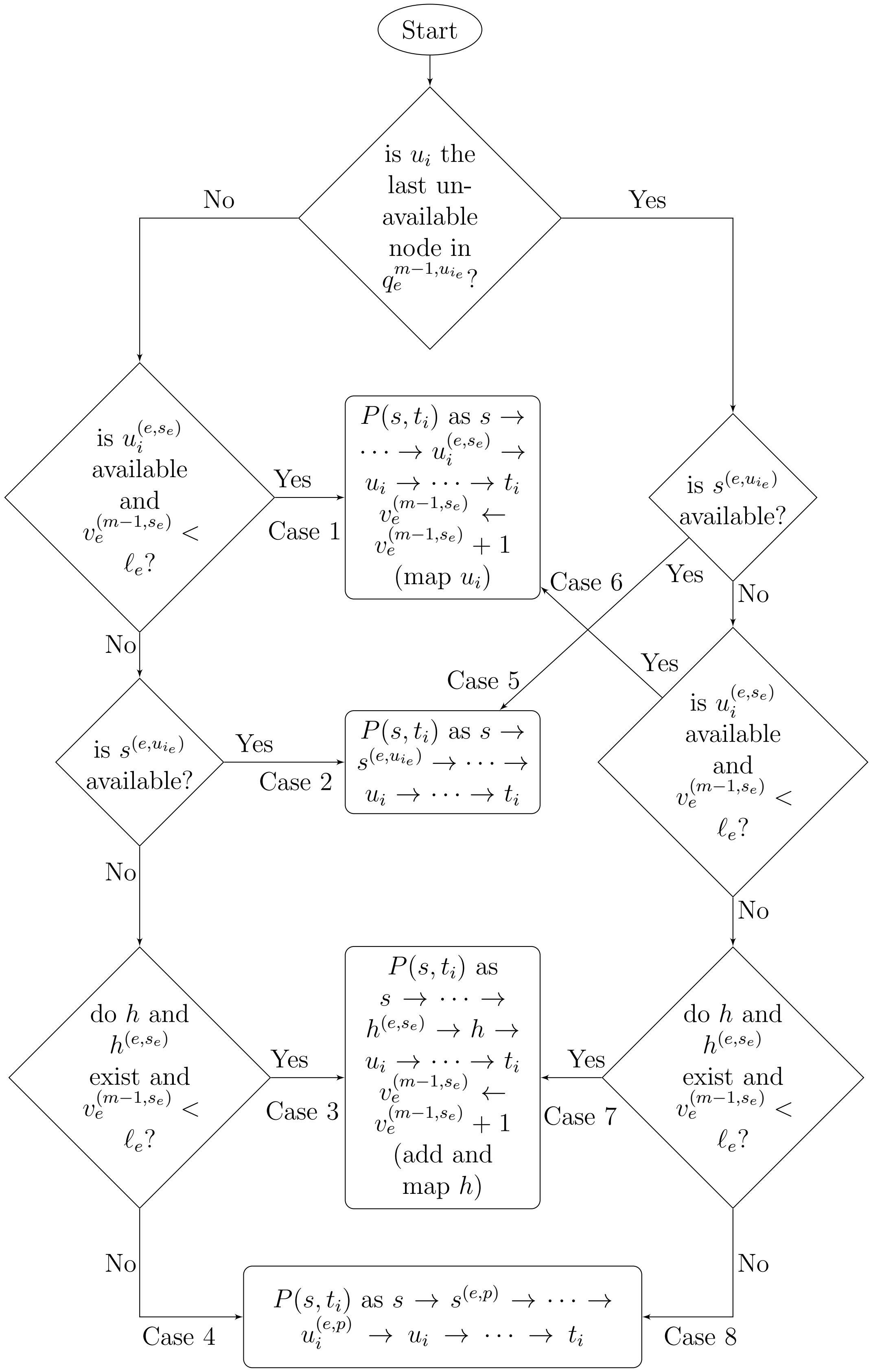


Figure 2.12: All cases of Step 4 of Algorithm 2

$s_0) \rightarrow (s_{m-1} \ldots s_{e+1} u_{i_e} s_{e-1} \ldots s_0) \rightarrow \cdots \rightarrow (u_{i_{m-1}} \ldots u_{i_0}) \rightarrow \cdots \rightarrow (t_{i_{m-1}} \ldots t_{i_0})$, such that the path from $s^{(e,u_{i_e})}$ to u_i is within the same subcube $q_e^{m-1,u_{i_e}}$.

For example in Figure 2.11b where $s = (000)$, $e = 1$, and $s_e = 0$, consider the unavailable node $u_i = (021)$. Note the following: 1) this unavailable node is not the last unavailable node in the subcube $q_1^{2,2}$ because the unavailable node (022) is after $u_i = (021)$ in the ordered set (according to the sorting in Step 3), 2) its neighbor $u_i^{(e,s_e)} = (021)^{(1,0)} = (001)$ in the source's subcube is not available (a destination node), and 3) the source's neighbor $s^{(e,u_{i_e})} = (000)^{(1,2)} = (020)$ in $q_1^{2,2}$ is available. So, the algorithm constructs the path $P(000, 021)$ completely as $000 \rightarrow 020 \rightarrow 021$. Similarly, the algorithm constructs the path $P(000, 031)$ completely as $000 \rightarrow 030 \rightarrow 031$.

Case 3: Suppose the following: 1) u_i is not the last unavailable node in $q_e^{m-1,u_{i_e}}$, 2) $u_i^{(e,s_e)}$ is not available, 3) $s^{(e,u_{i_e})}$ is not available, 4) $v_e^{(m-1,s_e)} < \ell_e$, and 5) within the same subcube $q_e^{m-1,u_{i_e}}$, a neighbor of u_i is available and also the neighbors of this neighbor in the source's subcube q_e^{m-1,s_e} is also available.

The number of neighbors of u_i in its subcube is equal to ℓ_e. Let $h = (u_{i_{m-1}} \ldots u_{i_{e+1}} u_{i_e} u_{i_{e-1}} \ldots h_j \ldots u_{i_0})$ be any neighbor (available or unavailable) of u_i in the same subcube $q_e^{m-1,u_{i_e}}$ where $j \in \{0, 1, \ldots, e-1, e+1, \ldots, m-1\}$ and $h_j \in \{0, 1, \ldots, k_j - 1\}$. In this case, the algorithm performs the following actions:

1. Sort all neighbors of u_i in the same subcube $q_e^{m-1,u_{i_e}}$ in ascending order according to the Hamming distance from the source. This sorting is mainly for minimizing the path length by examining the availability of the closest neighbor to the source node.

2. For each node h starting from the top in this list, check whether h is available; and also check whether its neighbor in the source's subcube $h^{(e,s_e)} = (u_{i_{m-1}} \dots u_{i_{e+1}} s_e u_{i_{e-1}} \dots h_j \dots u_{i_0})$ is also available. Note that by assumption, at least one of u_i's neighbors satisfies the above condition.

3. Add h and $h^{(e,s_e)}$ to the path $P(s,t_i)$ as $s \to \cdots \to h^{(e,s_e)} \to h \to u_i \to \cdots \to t_i$. In the next iteration, $h^{(e,s_e)}$ is one of the unavailable nodes in U. If s and $h^{(e,s_e)}$ are neighbors, the path $P(s,t_i)$ has been connected as $s \to h^{(e,s_e)} \to h \to u_i \to \cdots \to t_i$. This action increases $v_e^{(m-1,s_e)}$ by one because $h^{(e,s_e)}$ becomes unavailable node in the source's subcube.

Later, an example is given for Case 7 and it is similar to Case 3.

Case 4: Suppose the following: 1) u_i is not the last unavailable node in $q_e^{m-1,u_{i_e}}$, 2) $u_i^{(e,s_e)}$ is not available or $v_e^{(m-1,s_e)} = \ell_e$, 3) $s^{(e,u_{i_e})}$ is not available, and 4) for each neighbor h of u_i, either h is not available or $h^{(e,s_e)}$ is not available. In this case, there must exist another subcube (other than q_e^{m-1,s_e}) that does not have any unavailable nodes (to be proved in Section 2.3.2). Let $q_e^{m-1,p}$ be this subcube such that $p \in \{0, 1, \dots, s_e - 1, s_e + 1, \dots, k_e - 1\}$. Let $s^{(e,p)} = (s_{m-1} \dots s_{e+1} p s_{e-1} \dots s_0)$ be the source's neighbor in the this subcube. Let $u_i^{(e,p)} = (u_{i_{m-1}} \dots u_{i_{e+1}} p u_{i_{e-1}} \dots u_{i_0})$ be the neighbor of u_i in the this subcube. In this case, the algorithm constructs the path $P(s,t_i)$ completely as $(s_{m-1} \dots s_0) \to (s_{m-1} \dots s_{e+1} p s_{e-1} \dots s_0) \to \cdots \to (u_{i_{m-1}} \dots u_{i_{e+1}} p u_{i_{e-1}} \dots u_{i_0}) \to (u_{i_{m-1}} \dots u_{i_0}) \to \cdots \to (t_{i_{m-1}} \dots t_{i_0})$ such that the path from $s^{(e,p)}$ to $u_i^{(e,p)}$ is within the subcube $q_e^{m-1,p}$ that has no unavailable nodes.

Later, an example is given for Case 8 and it is similar to Case 4.

Case 5: Suppose the following: 1) u_i *is* the last unavailable node in $q_e^{m-1,u_{ie}}$, and 2) $s^{(e,u_{ie})}$ is available. In this case, the algorithm performs same actions as in Case 2. It constructs the path $P(s,t_i)$ completely through the source's neighbor $s^{(e,u_{ie})}$ in the subcube that has u_i as $s \to s^{(e,u_{ie})} \to \cdots \to u_i \to \cdots \to t_i$. Please see the example given for Case 2 which is similar to Case 5.

Case 6: Suppose the following: 1) u_i is the last unavailable node in $q_e^{m-1,u_{ie}}$, 2) $s^{(e,u_{ie})}$ is not available, 3) $u_i^{(e,s_e)}$ is available, and 4) $v_e^{(m-1,s_e)} < \ell_e$. In this case, the algorithm performs same actions as in Case 1. It maps u_i by adding its neighbor $u_i^{(e,s_e)}$ in q_e^{m-1,s_e} to the path $P(s,t_i)$ as $s \to \cdots \to u_i^{(e,s_e)} \to u_i \to \cdots \to t_i$. In the next iteration, $u_i^{(e,s_e)}$ is one of the unavailable nodes in U.

Case 7: Suppose the following: 1) u_i is the last unavailable node in $q_e^{m-1,u_{ie}}$, 2) $s^{(e,u_{ie})}$ is not available, 3) $u_i^{(e,s_e)}$ is not available, 4) $v_e^{(m-1,s_e)} < \ell_e$, and 5) a neighbor h of u_i within the same subcube $q_e^{m-1,u_{ie}}$ and h's neighbor $h^{(e,s_e)}$ are both available. In this case, the algorithm performs same actions as in Case 3. It adds h and $h^{(e,s_e)}$ to the path $P(s,t_i)$ as $s \to \cdots \to h^{(e,s_e)} \to h \to u_i \to \cdots \to t_i$. In the next iteration, $h^{(e,s_e)}$ is one of the unavailable nodes in U.

For example in Figure 2.11b where $s = (000)$, $e = 1$, and $s_e = 0$, consider the unavailable node $u_i = (022)$. Note the following: 1) this unavailable node is the last unavailable node in $q_1^{2,2}$, 2) the source's neighbor $s^{(e,u_{ie})} = (000)^{(1,2)} = (020)$ in $q_1^{2,2}$ has been used by the path $P(000,021)$, 3) the neighbor of (022) in the source's subcube $q_1^{2,0}$, which is $u_i^{(e,s_e)} = (022)^{(1,0)} = (002)$, is not available (a destination node), 4) $v_1^{(2,0)} = 3 < \ell_1 = 4$, and 5) there exists a neighbor h of (022) in $q_1^{2,2}$ that is available and its neighbor $h^{(e,s_e)} = h^{(1,0)}$ in $q_1^{2,0}$ is also available. In this case, the algorithm performs the following actions:

1. Sort all neighbors of $u_i = (022)$ in $q_1^{2,2}$ in ascending order according to the Hamming distance from the source (000). The resultant ordered set is $\langle 020, 021, 122, 222 \rangle$.

2. For each node h starting from the top of this list, check the availability of h and its neighbor $h^{(1,0)}$. (020) and (021) are not available. (122) is available but its neighbor $(122)^{(1,0)} = (102)$ in $q_1^{2,0}$ is unavailable (a destination node). (222) and its neighbor $(222)^{(1,0)} = (202)$ are both available, so we add them to the path in the next step.

3. Add $h = (222)$ and $h^{(1,0)} = (202)$ to the path $P(000, 022)$ as $000 \to \cdots \to 202 \to 222 \to 022$. In the next iteration, $h^{(1,0)} = (202)$ is one of the unavailable nodes in U. This action increases $v_1^{(2,0)}$ by one. So, $v_1^{(2,0)} = 4$.

Since at this point $v_1^{(2,0)} = \ell_1 = 4$, any further mapping is not possible.

Case 8: Suppose the following: 1) u_i is the last unavailable node in $q_e^{m-1,u_{i_e}}$, 2) $s^{(e,u_{i_e})}$ is not available, 3) $u_i^{(e,s_e)}$ is not available or $v_e^{(m-1,s_e)} = \ell_e$, and 4) for each neighbor h of u_i within the same subcube $q_e^{m-1,u_{i_e}}$, either h is not available or $h^{(e,s_e)}$ is not available. In this case, there must exist another subcube $q_e^{m-1,p}$ (other than q_e^{m-1,s_e}) that does not have any unavailable nodes (to be proved in Section 2.3.2). The algorithm performs same actions as in

Case 4. It constructs the path $P(s, t_i)$ completely through the subcube $q_e^{m-1,p}$ that has no unavailable nodes as $s \to s^{(e,p)} \to \cdots \to u_i^{(e,p)} \to u_i \to \cdots \to t_i$. For example in Figure 2.11b where $s = (000)$, $e = 1$, and $s_e = 0$, consider the unavailable node $u_i = (032)$. Note the following: 1) this unavailable node is the last unavailable node in $q_1^{2,3}$, 2) the source's neighbor $s^{(e,u_{i_e})} = (000)^{(1,3)} = (030)$ in $q_1^{2,3}$ has been used by the path $P(000, 031)$, and 3) $v_1^{(2,0)} = \ell_1 = 4$. In this case note that, the subcube $q_1^{2,1}$ (where $p = 1$) does not have any unavailable nodes. The algorithm constructs the path $P(000, 032)$ completely as $000 \to 010 \to 012 \to 032$ where $s^{(e,p)} = (000)^{(1,1)} = (010)$ and $u_i^{(e,p)} = (032)^{(1,1)} = (012)$.

Step 5 (Iterate)

Unless all destination nodes in T have been reached, go to Step 2.

For example in Figure 2.11b, the algorithm constructs the following paths during the next iteration ($j = 2$):

- $P(000, 102)$ as $000 \to 100 \to 102$.
- $P(000, 022)$ as $000 \to 200 \to 202 \to 222 \to 022$ (completing the path from the previous iteration).

Example 2.3.1 provides a complete example of Algorithm 2.

Example 2.3.1. *Consider the GH $Q^4_{5,3,2,4}$. The node degree in this GH is $\ell = 10$. Let the source node be $s = (0000)$ and the set of the ten distinct destination nodes be $T = \{0011, 1102, 0212, 1013, 1202, 2012, 0210, 2113, 0113, 1111\}$. Algorithm 2 finds a set of NDP from s to each destination node in T as follows:*

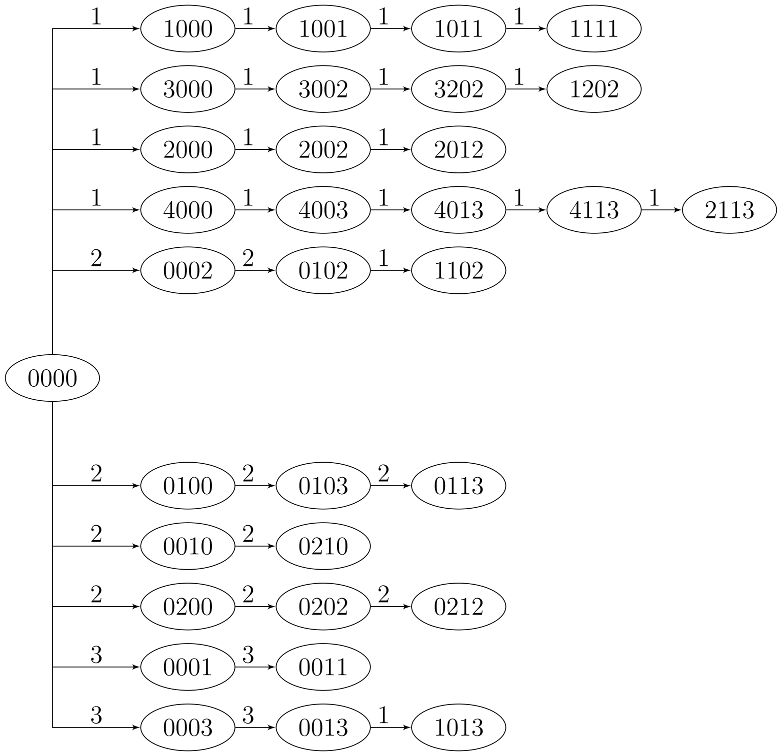


Figure 2.13: Iteration-wise of all NDP in Example 2.3.1

- *Step 1 (Reach the source's neighbors): All destination nodes in T are not neighbors of the source.*

- ***1**st **Iteration** ($j = 1, m = n = 4$):*

  - *Step 2 (Partition $Q^4_{5,3,2,4}$):*

    * $e = \arg\max\big\{\{v_0^{(3,0)} = 1 | v_0^{(3,0)} = 1, v_0^{(3,1)} = 2, v_0^{(3,2)} = 4, v_0^{(3,3)} = 3 \leq \ell_0 = 7\}, \{v_1^{(3,0)} = 2 | v_1^{(3,0)} = 2, v_1^{(3,1)} = 8 \leq \ell_1 = 9\}, \{v_2^{(3,0)} = 3 | v_2^{(3,0)} = 3, v_2^{(3,1)} = 4, v_2^{(3,2)} = 3 \leq \ell_2 = 8\}, \{v_3^{(3,0)} = 4 | v_3^{(3,0)} = 4, v_3^{(3,1)} = 4, v_3^{(3,2)} = 2, v_3^{(3,3)} = 0, v_3^{(3,4)} = 0 \leq \ell_3 = 6\}\big\} = 3$

    * *Partitions:*

      · $q_3^{3,0}$ *has* $\{0011, 0212, 0210, 0113\}$.

- $q_3^{3,1}$ *has* $\{1102, 1013, 1202, 1111\}$.
- $q_3^{3,2}$ *has* $\{2012, 2113\}$.
- $q_3^{3,3}$ *has no unavailable nodes.*
- $q_3^{3,4}$ *has no unavailable nodes.*

– *Step 3 (Sort the unavailable nodes that are not in* $q_3^{3,0}$*):*

  * $\langle 1013, 1102, 1111, 1202, 2012, 2113\rangle$

– *Step 4 (Construct the path completely or map):*

  * $P(0000, 1013)$ *is* $0000 \to \cdots \to 0013 \to 1013$ *(Case 1, map* (1013)*)*
    - $v_3^{(3,0)} \leftarrow v_3^{(3,0)} + 1 = 5$
  * $P(0000, 1102)$ *is* $0000 \to \cdots \to 0102 \to 1102$ *(Case 1, map* (1102)*)*
    - $v_3^{(3,0)} \leftarrow v_3^{(3,0)} + 1 = 6$ *(*$v_3^{(3,0)} = \ell_3$*, can't map any more)*
  * $P(0000, 1111)$ *is* $0000 \to 1000 \to 1001 \to 1011 \to 1111$ *(Case 2, construct the path completely)*
  * $P(0000, 1202)$ *is* $0000 \to 3000 \to 3002 \to 3202 \to 1202$ *(Case 4, construct the path completely through the subcube* $q_3^{3,3}$ *that has no unavailable nodes)*
  * $P(0000, 2012)$ *is* $0000 \to 2000 \to 2002 \to 2012$ *(Case 2, construct the path completely)*
  * $P(0000, 2113)$ *is* $0000 \to 4000 \to 4003 \to 4013 \to 4113 \to 2113$ *(Case 4, construct the path completely through the subcube* $q_3^{3,4}$ *that has no unavailable nodes)*

- ***2nd Iteration (*** $j = 2, m = 3$ ***):***

  – *Step 2 (Partition* $q_3^{3,0}$*):*

  * *The set of unavailable nodes in* $q_3^{3,0}$ *is* $\{0011, 0212, 0210, 0113, 0013, 0102\}$.
  * $e = \arg\max\{\{v_0^{(2,0)} = 1 | v_0^{(2,0)} = 1, v_0^{(2,1)} = 1, v_0^{(2,2)} = 2, v_0^{(2,3)} = 2 \leq \ell_0 = 3\}, \{v_1^{(2,0)} = 1 | v_1^{(2,0)} = 1, v_1^{(2,1)} = 5 \leq \ell_1 = 5\}, \{v_2^{(2,0)} = 2 | v_2^{(2,0)} = 2, v_2^{(2,1)} = 2, v_2^{(2,2)} = 2 \leq \ell_2 = 4\}\} = 2$
  * *Partitions:*
    · $q_2^{2,0}$ *has* $\{0011, 0013\}$.
    · $q_2^{2,1}$ *has* $\{0113, 0102\}$.
    · $q_2^{2,2}$ *has* $\{0212, 0210\}$.

- *Step 3 (Sort the unavailable nodes that are not in* $q_2^{2,0}$*):*
  * $\langle 0102, 0113, 0210, 0212 \rangle$

- *Step 4 (Construct the path completely or map):*
  * $P(0000, 1102)$ *is* $0000 \to 0002 \to 0102 \to 1102$ *(Case 1, map* (0102)*)*
    · $v_2^{(2,0)} \leftarrow v_2^{(2,0)} + 1 = 3$
  * $P(0000, 0113)$ *is* $0000 \to 0100 \to 0103 \to 0113$ *(Case 5, construct the path completely)*
  * $P(0000, 0210)$ *is* $0000 \to 0010 \to 0210$ *(Case 1, map* (0210)*)*
    · $v_2^{(2,0)} \leftarrow v_2^{(2,0)} + 1 = 4$ *(*$v_2^{(2,0)} = \ell_2$*, can't map any more)*
  * $P(0000, 0212)$ *is* $0000 \to 0200 \to 0202 \to 0212$ *(Case 5, construct the path completely)*

• ***3rd Iteration (*$j = 3, m = 2$*):***

- *Step 2 (Partition* $q_2^{2,0}$*):*

    * *The set of unavailable nodes in* $q_2^{2,0}$ *is* $\{0011, 0013, 0002, 0010\}$.
    * $e = \arg\max\big\{\{v_0^{(1,0)} = 1 | v_0^{(1,0)} = 1, v_0^{(1,1)} = 1, v_0^{(1,2)} = 1, v_0^{(1,3)} = 1 \leq \ell_0 = 1\}, \{v_1^{(1,0)} = 1 | v_1^{(1,0)} = 1, v_1^{(1,1)} = 3 \leq \ell_1 = 3\}\big\} = 0$
    * *Partitions:*
        · $q_0^{1,0}$ *has* $\{0010\}$.
        · $q_0^{1,1}$ *has* $\{0011\}$.
        · $q_0^{1,2}$ *has* $\{0002\}$.
        · $q_0^{1,2}$ *has* $\{0013\}$.
  – *Step 3 (Sort the unavailable nodes that are not in* $q_0^{1,0}$*):*
    * $\langle 0011, 0013 \rangle$
  – *Step 4 (Construct the path completely or map):*
    * $P(0000, 0011)$ *is* $0000 \rightarrow 0001 \rightarrow 0011$ *(Case 5, construct the path completely)*
    * $P(0000, 1013)$ *is* $0000 \rightarrow 0003 \rightarrow 0013 \rightarrow 1013$ *(Case 5, construct the path completely)*

All destination nodes in T *have been reached using a set of NDP. All paths are shown in Figure 2.13. Each link was constructed during the iteration number above it. Note how Algorithm 2 constructs the path (during more than one iteration) starting from the destination node by adding the internal nodes until the path is completely connected with the source node. For example,* $P(0000, 1013)$ *had to wait until the* 3^{rd} *iteration even its construction started during the* 1^{st} *iteration.*

■

	$m-1$		$m-2$		$\ldots$	1		0	
u_1	$m-1$		$m-2$			1		$\neq 0$	$\|U\| - v_0^{(m-1,0)}$
u_2	$m-1$		$m-2$			1		$\neq 0$	
.	$m-1$		$m-2$			$\neq 1$	$\|U\| - v_1^{(m-1,1)}$	0	
.	$m-1$		$m-2$			$\neq 1$		0	
.	.		.			.		.	
.	.		.			.		.	
.	.		.		$\ldots$	.		.	
.	$m-1$		$\neq (m-2)$	$\|U\| - v_{m-2}^{(m-1,m-2)}$		1		0	
.	$m-1$		$\neq (m-2)$			1		0	
.	$\neq (m-1)$	$\|U\| - v_{m-1}^{(m-1,m-1)}$	$m-2$			1		0	
$u_{\|U\|}$	$\neq (m-1)$		$m-2$			1		0	

Figure 2.14: Addresses of all unavailable nodes

2.3.2 Correctness of Algorithm 2

Theorem 2.3.3 proves the correctness of Algorithm 2. Theorem 2.3.1 and Theorem 2.3.2 are needed to prove Theorem 2.3.3.

Theorem 2.3.1. *In a GH $Q^m_{k_{m-1},\ldots,k_0}$, given any source node $s = (s_{m-1}\ldots s_0)$ and a set of distinct unavailable nodes $U = \{u_i = (u_{i_{m-1}}\ldots u_{i_0}) | 1 \leq i \leq \sum_{j=0}^{m-1}(k_j - 1)\}$, $Q^m_{k_{m-1},\ldots,k_0}$ can be partitioned along a dimension, say $e \in \{0, 1, \ldots, m-1\}$, such that the number $v_e^{(m-1,x)}$ of unavailable nodes in the subcube $q_e^{m-1,x}$ is at most $\ell_e = \sum_{j=0, j\neq e}^{m-1}(k_j - 1)$ for all $x \in \{0, 1, \ldots, (k_e - 1)\}$.*

Proof. We prove the theorem by contradiction. Assume that the statement is false. That means the number of unavailable nodes $v_e^{(m-1,x)}$ in $q_e^{m-1,x}$ for some $x \in \{0, 1, \ldots, (k_e - 1)\}$ is strictly greater than ℓ_e for all $e = 0, 1, \ldots, m-1$. Consider the e-th coordinate u_{i_e} of each unavailable node $u_i \in U$. Clearly, the number of unavailable nodes that has the value x in the e-th coordinate is equal to $v_e^{(m-1,x)}$. So, the number of unavailable nodes that have value $\neq x$ in the e-th coordinate is equal to $|U| - v_e^{(m-1,x)}$ (see Figure 2.14). Since we have assumed $v_e^{(m-1,x)} > \ell_e$, this is equivalent to $|U| - v_e^{(m-1,x)} < |U| - \ell_e$ for all $e = 0, 1, \ldots, m-1$. As shown in Figure 2.14, the number of distinct unavailable nodes is maximum when the '$\neq x$' elements in each dimension of the unavailable nodes addresses do not overlap. Thus, the maximum number of unavailable nodes is $\sum_{e=0}^{m-1}(|U|-v_e^{(m-1,x)})$. Since $|U| - v_e^{(m-1,x)} < |U| - \ell_e$, this is equivalent to $\sum_{e=0}^{m-1}(|U| - v_e^{(m-1,x)}) < \sum_{e=0}^{m-1}(|U| - \ell_e) = \sum_{e=0}^{m-1}(k_e - 1) = |U|$. This is a contradiction since it is assumed that the number of unavailable nodes is exactly equal to $|U|$. $\square$

Theorem 2.3.2 provides a set of maximum number of one-to-one NDP from a source to a destination. In [5], the authors have also given a method to find this set. However in their method, some nodes are in more than one path and so, the paths are not node disjoint. Thus, Theorem 2.3.2 is an improvement over their results. This theorem is useful in order to show that the paths construction in Case 2 and Case 5 of Step 4 of Algorithm 2 is possible.

Theorem 2.3.2. *In a $GHQ^m_{k_{m-1},\dots,k_0}$, suppose $a = (a_{m-1} \dots a_0)$ and $b = (b_{m-1} \dots b_0)$ are the source and destination nodes. There is a set of $l = \sum_{j=0}^{m-1}(k_j - 1)$ NDP as follows:*

1. *If $D_H(a, b) = m$, then there are m NDP of length m and $l-m$ NDP of length $m+1$.*

2. *If $D_H(a, b) = d < m$ and a and b differ in $i_1, i_2, \dots, i_d$ positions, then there are d NDP of length d, $\sum_{j=1}^{d}(k_{i_j} - 2)$ NDP of length $d + 1$, and $l - d - \sum_{j=1}^{d}(k_{i_j} - 2)$ NDP of length $d + 2$.*

Proof. **Case 1:** Suppose $D_H(a, b) = m$. In this case, there is a set of l NDP as follows:

1. The m NDP of length m are as follows (correct one digit at a time starting from the i-th digit going along right to left):

The i-th path, $i = 0, 1, \ldots, m-1$, is the following:

$(a_{n-1}a_{n-2} \ldots a_{i+1}a_ia_{i-1} \ldots a_0) \rightarrow (a_{n-1}a_{n-2} \ldots a_{i+1}b_ia_{i-1} \ldots a_0) \rightarrow$
$(a_{n-1}a_{n-2} \ldots b_{i+1}b_ia_{i-1} \ldots a_0) \rightarrow \cdots \rightarrow (b_{n-1}b_{n-2} \ldots b_{i+1}b_ia_{i-1} \ldots a_0) \rightarrow$
$(b_{n-1}b_{n-2} \ldots b_{i+1}b_ia_{i-1} \ldots a_1b_0) \rightarrow \cdots \rightarrow (b_{n-1}b_{n-2} \ldots b_{i+1}b_ib_{i-1} \ldots b_1b_0)$.

2. The remaining $l - m$ NDP of length $m + 1$ are as follows:

   For each i-th digit, first change a_i to c_i where $c_i \neq b_i$, $c_i \neq a_i$, and $c_i \in \{0, 1, 2, \ldots, k_i - 1\}$. Then as before correct one digit at a time starting from digit $i + 1$ going from right to left in cyclic order and finally change c_i to b_i. These paths have path length of $m + 1$.

It can be easily verified that all these l paths are node disjoint. Example 2.3.2 provides an example of this case.

Example 2.3.2. *For example, suppose $m = 3$, $k_0 = k_1 = k_2 = 4$, $a = (021)$, and $b = (132)$. In this case $D_H(a, b) = 3$. A set of $l = 3 + 3 + 3 = 9$ NDP is as follows:*

1. The $m = 3$ NDP of length $m = 3$ are (correct one digit at a time going from right to left in cyclic order):

   i) $021 \rightarrow 022 \rightarrow 032 \rightarrow 132$

   ii) $021 \rightarrow 031 \rightarrow 131 \rightarrow 132$

   iii) $021 \rightarrow 121 \rightarrow 122 \rightarrow 132$

2. The $l - m = 9 - 3 = 6$ NDP of length $m + 1 = 3 + 1 = 4$ are:

   i) $021 \rightarrow 020 \rightarrow 030 \rightarrow 130 \rightarrow 132$

   ii) $021 \rightarrow 023 \rightarrow 033 \rightarrow 133 \rightarrow 132$

iii) $021 \to 001 \to 101 \to 102 \to 132$

iv) $021 \to 011 \to 111 \to 112 \to 132$

v) $021 \to 221 \to 222 \to 232 \to 132$

vi) $021 \to 321 \to 322 \to 332 \to 132$

Case 2: Suppose $D_H(a, b) = d < m$. In this case, there is a set of l NDP as follows:

1. The d NDP of length d can be constructed by correcting one digit at a time, going from right to left in cyclic order. The i-th path, $i = 1, 2, \ldots, d$, starts with correcting the i-th digit in which they differ.

2. The $\sum_{j=1}^{d}(k_{i_j} - 2)$ NDP of length $d + 1$ can be constructed as follows:

   Suppose $i_1, i_2, \ldots, i_d$ are the positions in which $a_{i_j} \neq b_{i_j}$ for $j = 1, 2, \ldots,$ d. Change a_{i_j} to c_{i_j} where $c_{i_j} \neq a_{i_j}$, $c_{i_j} \neq b_{i_j}$, and $c_{i_j} \in \{0, 1, \ldots, k_{i_j} - 1\}$. First go from node a to the node a' where the node values are same as a except the i_j-th digit of a' is c_{i_j}. Then correct one digit at a time (i.e. a_t to b_t whenever the digits differ) going from right to left in cyclic order. After correcting all other digits, change c_{i_j} to b_{i_j}.

3. The $l - d - \sum_{j=1}^{d}(k_{i_j} - 2)$ NDP of length $d + 2$ are as follows:

   Suppose $a_{i_j} = b_{i_j}$ for $j = 1, 2, \ldots, m - d$. Change a_{i_j} to c_{i_j} where $c_{i_j} \neq a_{i_j}$ and $c_{i_j} \in \{0, 1, \ldots, k_{i_j} - 1\}$. In other words, go from a to a' where the node values are same as a except the i_j-th digit of a' is c_{i_j}. From a', construct the path by correcting one digit at a time where a and b differ and finally correcting c_{i_j} to b_{i_j}.

It can be easily verified that all those l paths are node disjoint. Example 2.3.3 provides an example of this case.

Example 2.3.3. *For example, suppose* $m = 3$, $k_0 = k_1 = k_2 = 4$, $a = (021)$, *and* $b = (032)$. *In this case* $D_H(a, b) = d = 2 < m = 3$. *A set of* $l = 3 + 3 + 3 = 9$ *NDP is as follows:*

1. *The* $d = 2$ *NDP of length* $d = 2$ *are:*

   i) $021 \to 022 \to 032$

   ii) $021 \to 031 \to 032$

2. *The* $\sum_{j=1}^{d=2}(k_{i_j} - 2) = (k_0 - 2) + (k_1 - 2) = 4$ *NDP of length* $d + 1 = 2 + 1 = 3$ *are:*

   i) $021 \to 020 \to 030 \to 032$

   ii) $021 \to 023 \to 033 \to 032$

   iii) $021 \to 001 \to 002 \to 032$

   iv) $021 \to 011 \to 012 \to 032$

3. *The* $l - d - \sum_{j=1}^{d}(k_{i_j} - 2) = 9 - 2 - 4 = 3$ *NDP of length* $d + 2 = 4$ *are:*

   i) $021 \to 121 \to 122 \to 132 \to 032$

   ii) $021 \to 221 \to 222 \to 232 \to 032$

   iii) $021 \to 321 \to 322 \to 332 \to 032$

$\square$

Theorem 2.3.3. *In a GH* $Q^n_{k_{n-1},\ldots,k_0}$, *given any source node* $s = (s_{n-1} \ldots s_0)$ *and a set of distinct destination nodes* $T = \{t_i = (t_{i_{n-1}} \ldots t_{i_0}) | 1 \le i \le \ell\}$ *such that* $s \notin T$ *and* $\ell = \sum_{j=0}^{n-1}(k_j - 1)$, *Algorithm 2 always finds a set of NDP* $\mathbb{P}(s, T)$ *with path lengths at most* $2n - 1$.

Proof. Since the GH is symmetric network, without loss of generality, assume that the source node s is $(00\ldots0)$.

Note that if a destination node t_i is at distance one, then the path $P(00\ldots0, t_i)$, by Step 1 of Algorithm 2, is $(00\ldots0) \rightarrow (00\ldots t_{i_j}\ldots0)$, $j \in \{0, 1, \ldots, n-1\}$, and its length is equal to one. Assume that the destination node t_i is at distance more than one. Let $\ell^{(n)} = \sum_{j=0}^{n-1}(k_j - 1)$ be the number of destination nodes in the n-dimensional GH. $\ell^{(n)}$ is also equal to the node degree in the n-dimensional GH. We prove that Algorithm 2 finds a set of NDP to $\ell^{(n)}$ distinct destination nodes with path lengths at most $2n - 1$ by induction on the dimension n of the GH $Q^n_{k_{n-1},\ldots,k_0}$.

Base Case: When $n = 2$, Algorithm 2 is equivalent to Algorithm 1. In this case as proved in Theorem 2.2.1, there are $k_0 + k_1 - 2$ NDP with lengths at most $2 \times 2 - 1 = 3$ in a GH $Q^2_{k_1,k_0}$.

Induction Step: Assume the hypothesis is true for $n - 1$. In other words, given a source node $s = (00\ldots0)$ and $\ell^{(n-1)} = \sum_{j=0}^{n-2}(k_j - 1)$ distinct destination nodes, Algorithm 2 finds a set of NDP from s to these $\ell^{(n-1)}$ destination nodes with path lengths at most $2(n-1) - 1 = 2n - 3$.

In Step 2 of Algorithm 2, the algorithm partitions the n-cube to k_e mutually disjoint $(n-1)$-cubes using the partitioning dimension e such that the node degree $\ell^{(n-1)}$ in all subcubes is equal to $\sum_{j=0}^{n-2}(k_j - 1)$ and the number of destination nodes in the subcube $q_e^{n-1,0}$ that has the source node is at most $\ell^{(n-1)}$. Theorem 2.3.1 proves that e always exists.

Let the set of distinct destination nodes in the n-cube be $U = \{u_i = (u_{i_{n-1}} \ldots u_{i_0}) | 1 \leq i \leq \ell^{(n)}\}$. Suppose that a destination node $u_i = (u_{i_{n-1}} \ldots u_{i_0}) \in U$ is in

$q_e^{n-1,0}$. Algorithm 2 makes the number $v_e^{(m-1,0)}$ of destination and internal nodes in $q_e^{n-1,0}$ exactly equal to $\ell^{(n-1)}$. Thus by induction hypothesis, there is a path $P(s, u_i)$ of length at most $2n-3$ to this destination node u_i. On the other hand, suppose u_i is in a subcube other than $q_e^{n-1,0}$. Then, there are two cases:

Case 1: Suppose the neighbor $(00\ldots0)^{(e,u_{i_e})} = (00\ldots0u_{i_e}0\ldots0)$ of the source node $(00\ldots0)$ in the subcube q_e^{n-1,u_i} that has u_i is available. Then, there are two subcases:

Case 1.1: Suppose $v_e^{(m-1,0)} = \ell^{(n-1)}$, meaning the mapping is not possible. Then by Case 2 or Case 5 of Step 4 of Algorithm 2, the path $P(00\ldots0, u_i)$ is $(00\ldots0) \to (00\ldots0u_{i_e}0\ldots0) \to \cdots \to (u_{i_{n-1}}\ldots u_{i_0})$. By Equation 2.2, the number of destination nodes in the subcube $q_e^{n-1,u_{i_e}}$ that has u_i is less than the node degree $\ell^{(n-1)}$ of this subcube. It follows that the previous path exists because according to Theorem 2.3.2 there are $\ell^{(n-1)}$ NDP from $(00\ldots0u_{i_e}0\ldots0)$ to u_i within the same subcube $q_e^{n-1,u_{i_e}}$. Its length is equal to $D_H(00\ldots0u_{i_e}0\ldots0, u_i) + 1 \leq 2n-1$.

Case 1.2: Suppose $v_e^{(m-1,0)} < \ell^{(n-1)}$, meaning the mapping is possible. Then, there are two subcases:

Case 1.2.1: Suppose u_i is the last destination node in $q_e^{n-1,u_{i_e}}$. Then by Case 2 of Step 4 of Algorithm 2, the path $P(00\ldots0, u_i)$ is the same as the path given in Case 1.1.

Case 1.2.2: Suppose u_i is not the last destination node in $q_e^{n-1,u_{i_e}}$. In this case the algorithm performs one of the following mappings:

1. By Case 1 of Step 4 of Algorithm 2, the algorithm maps u_i by adding $u_i^{(e,0)} = (u_{i_{n-1}}\ldots u_{i_{e+1}}0u_{i_{e-1}}\ldots u_{i_0})$ to $P(00\ldots0, u_i)$.

2. If the above mapping is not possible, then, by Case 3 of Step 4 of Algorithm 2, the algorithm maps the neighbor $h = (u_{i_{n-1}} \dots u_{i_{e+1}} u_{i_e} u_{i_{e-1}} \dots\ h_j \dots u_{i_0})$ of u_i in $q_e^{n-1,u_{i_e}}$ where $j \in \{0, 1, \dots, n-1\}$, $j \neq e$, and $h_j \in \{0, 1, \dots, k_j - 1\}$ by adding $h^{(e,0)} = (u_{i_{n-1}} \dots u_{i_{e+1}} 0 u_{i_{e-1}} \dots\ h_j \dots u_{i_0})$ to $P(00\dots0, u_i)$.

At least one of these two mappings is possible. We prove this by contradiction.

Assume that it is not possible to do any one of the above mappings. Since $v_e^{(m-1,0)} < \ell^{(n-1)}$, it follows $u_i^{(e,0)}$ and $h^{(e,0)}$ for all h such that h in $q_e^{n-1,u_{i_e}}$ and $D_H(h, u_i) = 1$ are not available. There are two cases:

1) Suppose u_i and $(00\dots0u_{i_e}0\dots0)$ are neighbors. Then, the number of neighbors of u_i in $q_e^{n-1,u_{i_e}}$ other than $(00\dots0u_{i_e}0\dots0)$ is equal to $\ell^{(n-1)} - 1$. It follows that $v_e^{(m-1,0)} = \ell^{(n-1)} - 1 + 1 = \ell^{(n-1)}$. This is a contradiction because it is assumed that $v_e^{(m-1,0)} < \ell^{(n-1)}$.

2) Suppose u_i and $(00\dots0u_{i_e}0\dots0)$ are not neighbors. Then, the number of neighbors of u_i in $q_e^{n-1,u_{i_e}}$ is equal to $\ell^{(n-1)}$. It follows that $v_e^{(m-1,0)} = \ell^{(n-1)} + 1$. This is a contradiction because it is assumed that $v_e^{(m-1,0)} < \ell^{(n-1)}$.

Thus, we have gotten a contradiction in both cases. So, at least one of the above mappings is possible. By the induction hypothesis, there is a path of length at most $2n-3$ to the mapped node. So, the path length to u_i is at most $2n - 3 + 2 = 2n - 1$.

Case 2: Suppose the neighbor $(00\ldots0)^{(e,u_{i_e})} = (00\ldots0u_{i_e}0\ldots0)$ of the source $(00\ldots0)$ in the subcube q_e^{n-1,u_i} that has u_i is not available. Then, there are two subcases:

Case 2.1: Suppose $v_e^{(m-1,0)} = \ell^{(n-1)}$. In this case (by Case 4 or Case 8 of Algorithm 2), the path $P(00\ldots0, u_i)$ is the following such that $q_e^{n-1,p}$ has no unavailable nodes:

$(00\ldots0) \to (00\ldots0p0\ldots0) \to \cdots \to (u_{i_{n-1}}\ldots u_{i_{e+1}}pu_{i_{e-1}}\ldots u_{i_0}) \to (u_{i_{n-1}}\ldots u_{i_0})$.

This subcube exists, which is proven below.

Assume that a subcube that has no unavailable nodes does not exist. This means there is at least one destination node in each of the subcubes other than the source's subcube. The number of these subcube is $k_e - 1 = \ell^{(n)} - \ell^{(n-1)}$. Thus, the number of destination nodes in these subcubes is at least $\ell^{(n)} - \ell^{(n-1)}$. As we have assumed that the source's subcube contains $\ell^{(n-1)}$ destination nodes. Thus, including u_i, the total number of destination nodes is equal to $\ell^{(n)} - \ell^{(n-1)} + \ell^{(n-1)} + 1 = \ell^{(n)} + 1$. This is a contradiction because it is assumed that the number of destination nodes is exactly equal to $\ell^{(n)}$. So, there must exist at least one subcube that has no unavailable nodes which means the above path exists and its length is equal to $D_H(00\ldots0p0\ldots0, u_{i_{n-1}}\ldots u_{i_{e+1}}pu_{i_{e-1}}\ldots u_{i_0}) + 1 + 1 \leq 2n - 1$.

Case 2.2: Suppose $v_e^{(m-1,0)} < \ell^{(n-1)}$. Then, this case is similar to Case 1.2.2.

This proves that Algorithm 2 is correct. $\square$

Since it is possible that Algorithm 2 maps along the same path during each iteration, Corollary 2.3.1 gives the upper and lower bounds of the length of any node disjoint path generated by Algorithm 2.

Corollary 2.3.1. *For any node disjoint path $P(s, t_i)$ in $\mathbb{P}(s, T)$ generated by Algorithm 2, the upper and lower bounds of path length are given in Inequation* (2.3)*.*

$$D_H(s, t_i) \leq |P(s, t_i)| \leq 2n - 1 \tag{2.3}$$

2.3.3 Time Complexity of Algorithm 2

In this section, we analyse the time complexity of Algorithm 2. For simplicity, assume that all dimensions have the same number of nodes. Let $k = k_0 = k_1 = \cdots = k_{n-1}$. If k_i's are not equal, we can replace k with k_{max} where $k_{max} = \max_{0 \leq i \leq (n-1)} \{k_i\}$. In the following we analyse the time complexity for each step of Algorithm 2 to find the overall time complexity:

1. In Step 1, Algorithm 2 constructs a path of length one to reach destination node at Hamming distance one. Since there are $\ell = n(k-1)$ destination nodes and Algorithm 2 checks each and every destination node, Step 1 takes $O(kn)$ time. Algorithm 2 performs this step one time.

2. In Step 2, Algorithm 2 partitions an m-cube into k subcubes of dimension $(m-1)$ where $m = n - (j-1)$ and j is the iteration counter. Since the partitioning is based on digit sets of the destination nodes, Algorithm 2 can use the bucket sort method. Thus, Step 2 takes $O(m\ell^{(m)})$ time in each iteration where $\ell^{(m)} = m(k-1)$ is the number of destination nodes in the

m-cube. So, the time complexity in all iterations is $\sum_{j=1}^{(n-1)} m\ell^{(m)} = (n)\ell^{(n)} + (n-1)\ell^{(n-1)} + \cdots + (2)\ell^{(2)} = (k-1)[n^2 + (n-1)^2 + (n-2)^2 + \cdots + 2^2] = O((k-1)n^3)$.

3. In Step 3, Algorithm 2 sorts all destination nodes that are not in the source's subcube. The number of these destination nodes is at most $\ell^{(m)}$. Since the sorting is based on digit sets of the destination nodes, Algorithm 2 can use the bucket sort method. Thus, Step 3 takes $O(m\ell^{(m)})$ time in each iteration. So, it also takes $O((k-1)n^3)$ time in all iterations.

4. In Step 4, for each destination node in all subcubes except the source's subcube, Algorithm 2 constructs a complete path to this destination node or adds one or two internal node(s) to this path according to the eight cases provided in Step 4 (see Section 2.3.1). Clearly, Case 3 and Case 7 are the most time consuming cases because they involve finding an available neighbor within the same subcube that its neighbor in the source's subcube is also available.

   The number of the destination nodes in all subcubes except the source's subcube is at most $\ell^{(m)}$. However, Algorithm 2 designates $(k-1)$ destination nodes to be reached through the source's immediate neighbor in each subcube (Case 2 or Case 5). So, these $(k-1)$ destination nodes do not follow Case 3 or Case 7. The remaining number of destination nodes that could follow Case 3 or Case 7 is $\ell^{(m)} - (k-1) = m(k-1) - (k-1) = (m-1)(k-1)$. The worst case occurs when these $(m-1)(k-1)$ destination nodes have distributed evenly among the $(k-1)$ subcubes. In this case, all these $(m-1)(k-1)$ destination nodes follow Case 3 or Case 7.

In these two cases, Algorithm 2 finds a neighbor of the destination node within the same subcube that is available and its neighbor in the source's subcube is also available. This involves the following operations:

(a) Sort all neighbors of the destination node within the same subcube according to the Hamming distance from the source to these neighbors. It can be seen that this Hamming distance is between the Hamming distance from the source to the destination node ± 2. So, we can assume that this sorting takes linear time.

(b) Check the availability of each neighbor of the destination node within the same subcube. If it is available, check the availability of its neighbor in the source's subcube. Since there are $\ell^{(m-1)} = (m-1)(k-1)$ neighbors of each destination node within the same subcube, this operation takes $O((k-1)(m-1))$ time for each destination node.

Thus, Step 4 takes $O([(k-1)(m-1)]^2)$ time in each iteration. So, the time complexity in all iterations is $\sum_{j=1}^{(n-1)}((k-1)(m-1))^2 = (k-1)^2[(n-1)^2 + (n-2)^2 + \cdots + 2^2] = O(k^2n^3)$.

From the above analysis, we can see that Step 4 is the most time consuming step. So, the overall time complexity of Algorithm 2 is $O(k^2n^3)$ where $k = k_0 = k_1 = \cdots = k_{n-1}$. If k_i's are not equal, the overall time complexity is $O({k_{max}}^2n^3)$ where $k_{max} = \max\limits_{0 \le i \le (n-1)} \{k_i\}$.

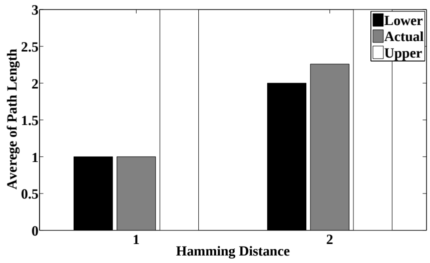
2.4 Simulation Results

In this section, we show the results of simulating the proposed algorithms. We mainly measure the path lengths and compare them to the Hamming distance $D_H(s, t_i)$ which is the shortest distance. Our simulation results show that all of the time our proposed algorithms give a set of node disjoint paths (NDP) and most of the time the length of the longest path is equal to $D_H(s, t_i) + 1$, which is less than the upper bound, $2n - 1$, as defined in Inequation (2.3).

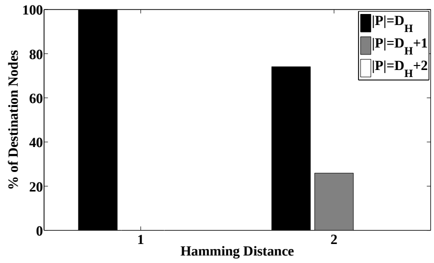
We ran a simulator of Algorithm 1 500 times using 80 nodes in each dimension ($k_0 = k_1 = 80$) which makes the total number of destination nodes equals $2 \times 79 = 158$. In each run, the simulator *randomly* generated these 158 destination nodes and the source node. It returned a set of NDP for each run. After taking the average of the path lengths for all runs, the results are shown in Figure 2.15.

Figure 2.15a shows the average of the path lengths as a function of the Hamming distance, along with the lower and upper bounds as defined in Inequation (2.1). For destination nodes at Hamming distance one (neighbors), Algorithm 1 reached all of them using paths of length one. This was expected because Algorithm 1 reaches all destination nodes at distance one (Step 1) before considering any other destination nodes. For destination nodes at Hamming distance two, the average of the path lengths equals about 2.25 which is very close to the shortest distance (the lower bound) $D_H(s, t_i)$.

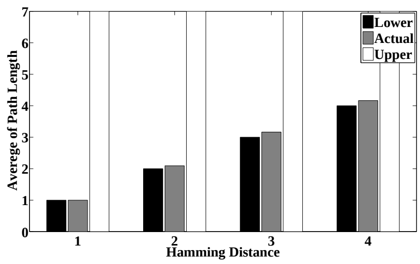
The path lengths average is close to the shortest distance because Algorithm 1 checks all possible paths of length two (Case 1 and Case 2 of Step 2 of Algorithm 1) before using a path of length three (Case 3 of Step 3 Algorithm 1). Figure 2.15b shows more details. It shows the percentage of destination nodes as a function of



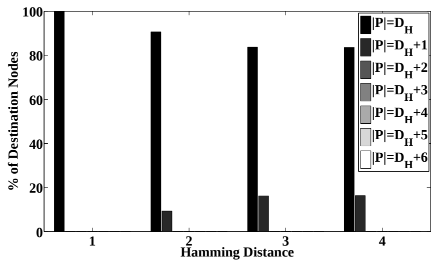
(a) Average of the path length



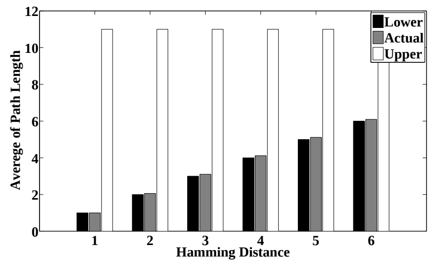
(b) Distribution of dest. nodes



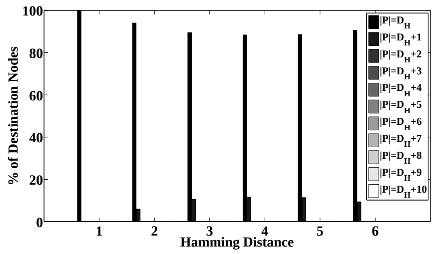
(c) Average of actual path length



(d) Distribution of dest. nodes



(e) Average of actual path length



(f) Distribution of dest. nodes

Figure 2.15: Simulation results: (a,b) Algorithm 1 ($n = 2$), (c,d) Algorithm 2 ($n = 4$), (e,f) Algorithm 2 ($n = 6$)

the Hamming distance for all possible lengths. In this figure, 100% of destination nodes at Hamming distance one have been reached by paths of length one. About 75% of destination nodes at Hamming distance two have been reached by paths of length two and the rest (about 25%) have been reached by paths of length three. So, the majority of destination nodes have been reached using the shortest paths. Thus, the path lengths average is close to the shortest distance.

We ran a simulator of Algorithm 2 to find a set of NDP for each of the following networks: 1) a four-dimensional ($n = 4$) GH with 30 nodes in each dimension ($k_0 = \cdots = k_3 = 30$) and $4 \times 29 = 116$ destination nodes, and 2) a six-dimensional ($n = 6$) GH with 8 nodes in each dimension ($k_0 = \cdots = k_5 = 8$) and $6 \times 7 = 42$ destination nodes. We ran the simulator 500 times for each network. For each one of the 1000 runs, the simulator returned a set of NDP. Figure 2.15 shows the results.

From these results, we note the following:

1. The actual path length in Figure 2.15c and Figure 2.15e is always between the lower and upper bounds as defined in Inequation (2.3).

2. The actual path length is always closer to the shortest distance (the lower bound).

3. The majority of destination nodes (80% to 90%) have been reached using the shortest paths as shown in Figure 2.15d and Figure 2.15f.

4. Most of the time the maximum length of a path $P(s, t_i)$ returned by Algorithm 2 is $D_H(s, t_i) + 1$.

Thus, the practical upper bound is $D_H(s, t_i) + 1$ which is less than the theoretical upper bound $2n - 1$. This is because Algorithm 2 minimizes the path lengths as much as possible by incorporating the following:

1. Algorithm 2 reaches all destination nodes at Hamming distance one before reaching all other nodes (Step 1).

2. Algorithm 2 checks the source's neighbor $s^{(e,t_{ie})}$ in the subcube that has the destination node t_i and the destination node's neighbor $t_i^{(e,s_e)}$ in the subcube

that has the source node s before using a path going through a neighbor h of t_i within the same subcube or a subcube that has no unavailable nodes.

3. In Case 3 and Case 7 of Step 4, Algorithm 2 chooses the neighbor h of the destination node t_i within the same subcube such that: 1) h is available, 2) the neighbor $h^{(e,s_e)}$ of h in the subcube that has the source node s is also available, and more importantly 3) among all the t_i neighbors which are in the same subcube as t_i, Algorithm 2 chooses the neighbor h such that its neighbor $h^{(e,s_e)}$ is available and also h has the smallest distance from the source.

Thus in practice, for any destination node t_i, the proposed algorithm finds a path of length at most $D_H(s, t_i) + 1$.

2.5 Conclusion

In this chapter we provide and prove some novel algorithms to find a set of the maximum number of one-to-many NDP from a source node to a set of destination nodes in the Generalized Hypercube interconnection networks. We show that the lower bound of each path is equal to the Hamming distance between the source node and that destination node, and the upper bound is equal to $2n - 1$ where n is the number of dimensions. In most cases, the proposed algorithms find a set of NDP with lengths at most one plus the Hamming distance between the source and destination nodes. We also show that the time complexity of the algorithm is $O({k_{max}}^2 n^3)$ where $k_{max} = \max_{0 \le i \le (n-1)} \{k_i\}$ and k_i is the number of nodes in dimension i.

Chapter 3: One-to-Many Node Disjoint Paths Routing in Dense Gaussian Networks*

In this chapter, an efficient constant time complexity algorithm that constructs node disjoint paths (NDP) from a single source node to the maximum number of destination nodes in dense Gaussian networks (DGNs) is given. Then, it is proved that this algorithm always returns a solution. Also, the lower and upper bounds of the sum of the NDP lengths are derived. Finally, via execution of the algorithm, it is shown that on the average the sum of lengths of NDP given by the algorithm is only about 10% more than the sum of the shortest paths lengths.

DGNs have significant topological advantages over torus networks in terms of diameter [31]. For example, there is a DGN with 400 nodes and diameter 14, whereas, any 2D toroidal network with 400 nodes will have a diameter of at least 20. So compare to torus networks, DGNs can accommodate more nodes with less communication latency and at the same time maintaining a regular grid-like structure. This makes DGNs attractive networks.

The rest of this chapter is organized as follows: Section 3.1 recalls several preliminaries about DGNs, Section 3.2 describes the proposed routing algorithm, Section 3.3 shows the algorithm execution results, and Section 3.4 concludes this chapter.

*This chapter has been accepted for publication in The Computer Journal.

3.1 Dense Gaussian Networks Preliminaries

Dense Gaussian Networks (DGNs) are defined in terms of Gaussian integers. The following subsections explain the Gaussian integers, describe DGNs, and formally define the one-to-many node disjoint paths (NDP) routing problem in these networks.

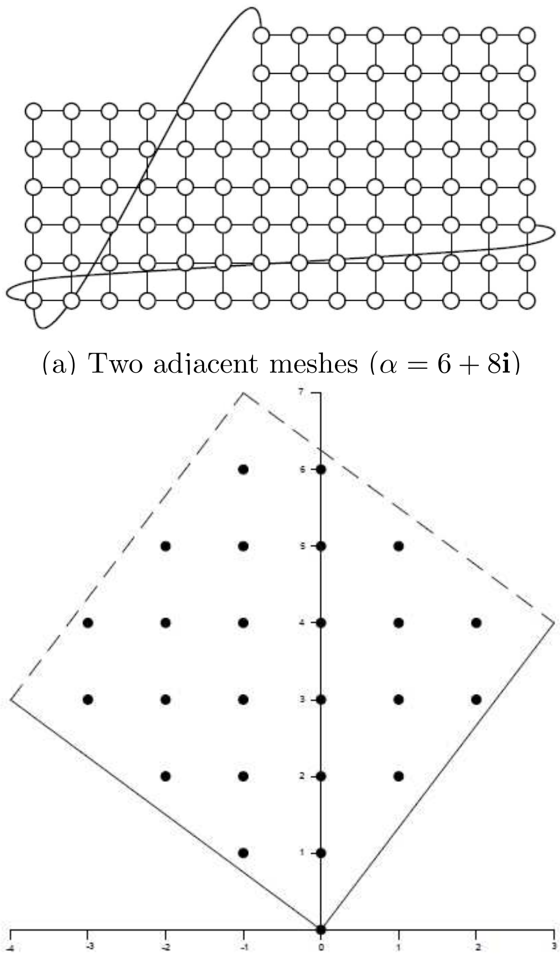
3.1.1 Gaussian Integers

A Gaussian integer is a complex number whose real and imaginary parts are both integers. The set of all Gaussian integers, $\mathbb{Z}[\mathbf{i}]$, is defined as $\{x + y\mathbf{i} | x, y \in \mathbb{Z}\}$ where $\mathbf{i} = \sqrt{-1}$.

The set $\mathbb{Z}[\mathbf{i}]$ is a Euclidean domain and the norm of a Gaussian integer $\omega = \omega_x + \omega_y\mathbf{i}$ is defined as [17]:

$$\mathcal{N}(\omega) = {\omega_x}^2 + {\omega_y}^2.$$

So, a Euclidean division algorithm for Gaussian integers exists. Let $\omega_1, \omega_2 \in \mathbb{Z}[\mathbf{i}]$ and $\omega_2 \neq 0$. Then, there exist $q, r \in \mathbb{Z}[\mathbf{i}]$ such that $\omega_1 = q\omega_2 + r$ and $\mathcal{N}(r) < \mathcal{N}(\omega_2)$. Let $\alpha = a + b\mathbf{i} \in \mathbb{Z}[\mathbf{i}]$ be nonzero where a and b are integers. Then, $\omega_1, \omega_2 \in \mathbb{Z}[\mathbf{i}]$ are congruent modulo α if there exists $\gamma \in \mathbb{Z}[\mathbf{i}]$ such that $\omega_2 - \omega_1 = \gamma\alpha$. Congruence and the Gaussian integers modulo α are denoted by $\omega_2 \equiv \omega_1 (\mathrm{mod}\ \alpha)$ and $\mathbb{Z}[\mathbf{i}]_\alpha$ respectively. The number of elements in $\mathbb{Z}[\mathbf{i}]_\alpha$ is equal to $\mathcal{N}(\alpha) = a^2 + b^2$ [17].



(a) Two adjacent meshes ($\alpha = 6 + 8\mathbf{i}$)

(b) Leaned square ($\alpha = 3 + 4\mathbf{i}$)

Figure 3.1: Different representations of Gaussian networks

3.1.2 Dense Gaussian Networks

DGNs are two-dimensional networks generated by Gaussian integers and these were first introduced in [31]. Let $\alpha \in \mathbb{Z}[\mathbf{i}]$ be nonzero. Each node's address in a DGN generated by α is a Gaussian integer that belongs to the Gaussian integers modulo α denoted by $\mathbb{Z}[\mathbf{i}]_\alpha$. So, the number of nodes in this DGN is equal to $\mathcal{N}(\alpha)$. These nodes can be represented in several ways. One representation is by placing the nodes on two adjacent square meshes (see Figure 3.1a) [31, 42]. Another repre-

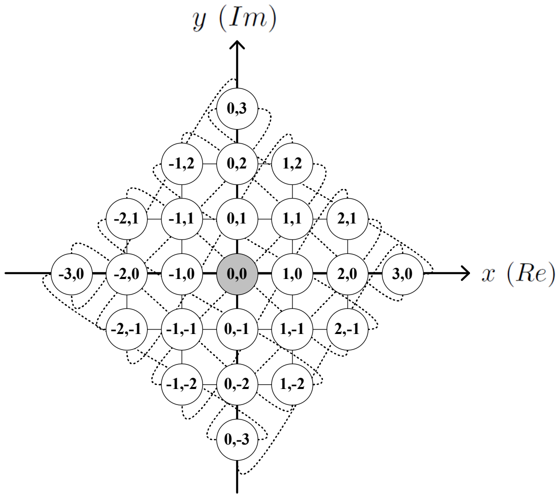


Figure 3.2: DGN G_3 ($\alpha = 3 + 4\mathbf{i}$)

sentation is by placing the nodes on a leaned square (see Figure 3.1b) [15, 42]. In this work, we use different representation which is explained in [33] (see Figure 3.2). In this representation, the nodes are placed on a two-dimensional Cartesian plan where the x-axis and y-axis represent the real and imaginary parts of each node respectively.

It is proved that for a given diameter $k \in \mathbb{Z}^+$, a DGN achieves the largest network size with $k^2 + (k+1)^2$ nodes when it is generated by $\alpha = k + (k+1)\mathbf{i}$ [31]. This network is referred as DGN and we find NDP in these DGNs. In this work, we assume the generator of the DGN is $\alpha = k + (k+1)\mathbf{i}$ and denote this DGN by G_k where k is the network diameter. Figure 3.2 shows the DGN G_3 generated by $\alpha = 3+4\mathbf{i}$. In this example, the number of nodes is equal to $\mathcal{N}(3+4\mathbf{i}) = 3^2+4^2 = 25$ and the diameter $k = 3$.

In the following, we explain the representation given in [33] in terms of the addressing, connectivity, diameter, degree, and shortest distance.

Addressing: Each node in the DGN generated by $\alpha = k + (k+1)\mathbf{i}$ is represented as $\omega = \omega_x + \omega_y\mathbf{i} \in \mathbb{Z}[\mathbf{i}]_\alpha$. For simplicity, we write $\omega = (\omega_x, \omega_y)$ to denote node ω in the network. The set of all nodes in G_k is $\{\omega = (\omega_x, \omega_y) \in \mathbb{Z} \times \mathbb{Z} \mid |\omega_x| + |\omega_y| \leq k\}$. In Figure 3.2, the 2-tuples inside each node are the addresses.

Connectivity: Two nodes $\omega_1, \omega_2 \in \mathbb{Z}[\mathbf{i}]_\alpha$ in G_k are connected (neighbors) if and only if $(\omega_1 - \omega_2) \equiv \pm 1, \pm\mathbf{i} \pmod{\alpha}$ where $\alpha = k + (k+1)\mathbf{i}$ is the generator of G_k. So, each node $\omega = \omega_x + \omega_y\mathbf{i} \in \mathbb{Z}[\mathbf{i}]_\alpha$ is connected to four neighbors:

1. the north neighbor $\omega^N = \omega_x + (\omega_y + 1)\mathbf{i} \pmod{\alpha}$,
2. the west neighbor $\omega^W = (\omega_x - 1) + \omega_y\mathbf{i} \pmod{\alpha}$,
3. the south neighbor $\omega^S = \omega_x + (\omega_y - 1)\mathbf{i} \pmod{\alpha}$, and
4. the east neighbor $\omega^E = (\omega_x + 1) + \omega_y\mathbf{i} \pmod{\alpha}$

where $\omega^N, \omega^W, \omega^S, \omega^E \in \mathbb{Z}[\mathbf{i}]_\alpha$.

The modulo function $(\mathrm{mod}\ \alpha)$ is used to build the wraparound links. Let $\beta = \beta_x + \beta_y\mathbf{i} \in \{\omega_x + (\omega_y + 1)\mathbf{i}, (\omega_x - 1) + \omega_y\mathbf{i}, \omega_x + (\omega_y - 1)\mathbf{i}, (\omega_x + 1) + \omega_y\mathbf{i}\}$ be one of the neighbors before applying the modulo function where $\beta \notin \mathbb{Z}[\mathbf{i}]_\alpha$. Then, the modulo function $\beta \pmod{\alpha}$ is given by the following [31]:

$$\beta \pmod{\alpha} = \begin{cases} \beta - \alpha & \text{if} \quad (\beta_x \geq 0) \wedge (\beta_y \geq 1) \\ \beta - \mathbf{i}\alpha & \text{if} \quad (\beta_x \leq -1) \wedge (\beta_y \geq 0) \\ \beta + \alpha & \text{if} \quad (\beta_x \leq 0) \wedge (\beta_y \leq -1) \\ \beta + \mathbf{i}\alpha & \text{if} \quad (\beta_x \geq 1) \wedge (\beta_y \leq 0) \end{cases} \tag{3.1}$$

In Figure 3.2, the dashed links are the wraparound links built using Equation 3.1 and these wraparound links always connect two boarder nodes (as defined in Definition 3.1.1 below) using Equation 3.1. For example, the south neighbor of $\omega = -2 - \mathbf{i}$ is $\omega^S = -2 - 2\mathbf{i} \pmod{3 + 4\mathbf{i}} = (-2 - 2\mathbf{i}) + (3 + 4\mathbf{i}) = 1 + 2\mathbf{i}$ where $\beta = -2 - 2\mathbf{i}$. Another example is that the north neighbor of $\omega = -2 + \mathbf{i}$ is $\omega^N = -2 + 2\mathbf{i} \pmod{3 + 4\mathbf{i}} = (-2 + 2\mathbf{i}) - \mathbf{i}\ (3 + 4\mathbf{i}) = (-2 + 2\mathbf{i}) + (4 - 3\mathbf{i}) = 2 - \mathbf{i}$ where $\beta = -2 + 2\mathbf{i}$.

Diameter: The diameter is the largest possible distance between any two nodes in a network. The diameter of G_k is equal to k [31]. For example in Figure 3.2, the diameter of G_3 is equal to three.

Degree: The node degree is the number of its neighbors. In DGNs, each node is adjacent to four other nodes. So, the node degree is equal to four for all nodes [31, 33].

Path: A path from node ω_1 to node ω_2 is denoted by $P(\omega_1, \omega_2) = \langle \omega_1, a_1, a_2, \ldots, a_{|P(\omega_1,\omega_2)|-1}, \omega_2 \rangle$ where $|P(\omega_1, \omega_2)|$ is the length of the path and each two consecutive nodes (e.g. ω_1 and a_1) along this path are neighbors. Sometimes, we write the path $P(\omega_1, \omega_2)$ as $\omega_1 \to a_1 \to a_2 \to \cdots \to \omega_2$.

Distance: The shortest distance between any two nodes $\omega_1, \omega_2 \in \mathbb{Z}[\mathbf{i}]_\alpha$ as defined in [31] is as follows:

$$D_\alpha(\omega_1, \omega_2) = \min\{|x| + |y| \mid (\omega_1 - \omega_2) \equiv (x + y\mathbf{i}) (\mathrm{mod}\ \alpha)\}.$$

For example in Figure 3.2, the shortest distance between $(0, 1)$ and $(1, 2)$ is

$$D_\alpha((0, 1), (1, 2)) = |0 - 1| + |1 - 2| = | - 1| + | - 1| = 2.$$

The length of the shortest path\paths from ω_1 to ω_2 equals\equal to the shortest distance $D_\alpha(\omega_1, \omega_2)$ between them. For example in Figure 3.2, one of the shortest paths between $(0, 0)$ and $(1, 1)$ is $(0, 0) \to (1, 0) \to (1, 1)$ of length

$D_\alpha((0,0),(1,1)) = 2$. An example of a longer path $P((0,0),(1,1))$ is $(0,0) \rightarrow (-1,0) \rightarrow (-1,1) \rightarrow (0,1) \rightarrow (1,1)$.

Since G_k is a Cayley graph with generator $\{1, -1, \mathbf{i}, -\mathbf{i}\}$, G_k is vertex symmetric [3]. So, the shortest distance between node $(0,0)$ and node $\omega = (\omega_x, \omega_y) \in \mathbb{Z}[\mathbf{i}]_\alpha$ is equal to ω's weight $W(\omega) \in \{0, 1, 2, \ldots, k\}$, which is defined as [31]:

$$W(\omega) = |\omega_x| + |\omega_y|. \tag{3.2}$$

For example in Figure 3.2, the weight of $(1,2)$ is $W((1,2)) = 3$ which is the shortest distance between $(0,0)$ and $(1,2)$.

Based on the weight, Definition 3.1.1 defines a border node.

Definition 3.1.1. *Let $\omega \in \mathbb{Z}[\mathbf{i}]_\alpha$ be any node. Then, ω is* a border node *if and only if $W(\omega) = k$ where k is the network diameter.*

For example in Figure 3.2, the nodes $(1,2)$, $(3,0)$, and $(-1,-2)$ are some of the border nodes.

The distance distribution of G_k gives the number of nodes $H(r)$ at distance $r \in \{0, 1, 2, \ldots, k\}$ from the $(0,0)$ node. This distribution is defined as follows [31]:

$$H(r) = \begin{cases} 1 & \text{if} \quad r = 0 \\ 4r & \text{if} \quad 1 \leq r \leq k \end{cases}$$

For example in G_3, $H(0) = 1$, $H(1) = 4$, $H(2) = 8$, and $H(3) = 12$.

Theorem 3.1.1 gives the shortest distance between node $(0,0)$ and any other node $\omega \in \mathbb{Z}[\mathbf{i}]_\alpha$ through exactly one wraparound link as a function of its weight $W(\omega)$. We use this theorem to calculate the length of some NDP.

Theorem 3.1.1. *In a DGN G_k where k is the network diameter, let $\omega \in \mathbb{Z}[\mathbf{i}]_\alpha$ be any node such that $\omega \neq (0,0)$. Then using one and only one wraparound link, the shortest distance $R(\omega)$ from node $(0,0)$ to node ω is given as follows:*

$$R(\omega) = 2k + 1 - W(\omega) \tag{3.3}$$

Proof. Since any wraparound link connects two border nodes, let a and b be the border nodes that are connected using the wraparound link in the path from node $(0,0)$ to node ω. It follows that $P((0,0),\omega)$ is $P((0,0),a) \to P(a,b) \to P(b,\omega)$. To find the lowest length $R(\omega)$ of this path , add $P(\omega,(0,0))$ to the end to be $P((0,0),a) \to P(a,b) \to P(b,\omega) \to P(\omega,(0,0))$. Since we want the length of the shortest path for each one of these paths, we know that:

1. $|P((0,0),a)| = W(a) = k$,

2. $|P(a,b)| = 1$ (because they are adjacent),

3. $|P(b,\omega)| + |P(\omega,(0,0))| = W(b) = k$, and

4. $P(\omega,(0,0)) = W(\omega)$.

It follows that $R(\omega) = 2k + 1 - W(\omega)$ □

In this work, sometimes we need to use a wraparound link to construct a path of the NDP. Theorem 3.1.1 is useful to calculate the extra number of hops

$$\delta(\omega) = R(\omega) - W(\omega) = 2k + 1 - 2W(\omega)$$

in order to use a wraparound link to reach ω. For example in Figure 3.2, the extra

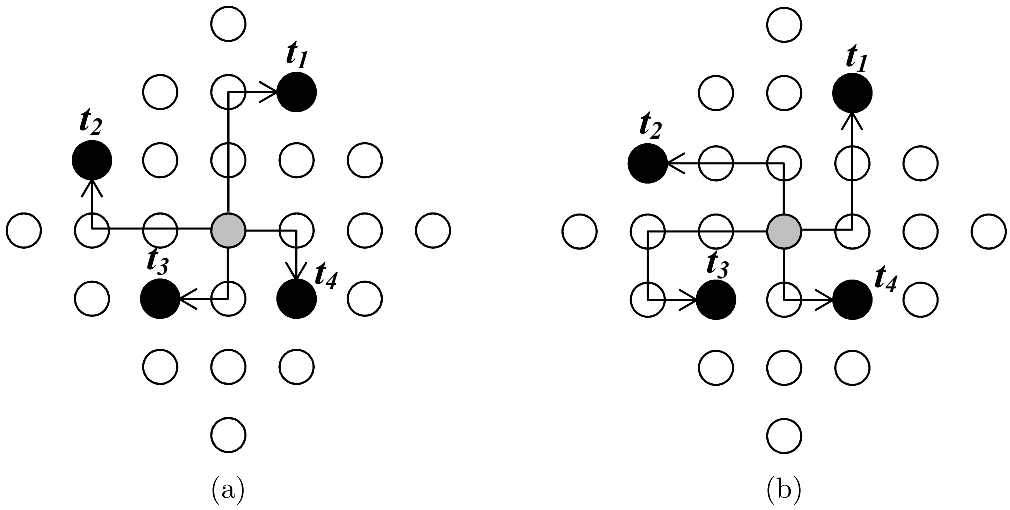


Figure 3.3: Different examples of NDP in G_3

number of hops in order to use a wraparound link from node $(0,0)$ to node $(1,1)$ equals $\delta((1,1)) = 2 \times 3 + 1 - 2 - 2 = 3$ hops.

In terms of the number of extra hops, the length of path $P((0,0),\omega)$ is given by

$$|P((0,0),\omega)| = W(\omega) + \delta(\omega).$$

One-to-Many NDP: As explained in the introduction, the one-to-many NDP in the DGNs connect the source node s with each destination node in $T = \{t_j = (t_{j_x}, t_{j_y}) | 1 \leq j \leq 4\}$ such that the disjointness condition is satisfied. Under this condition, the maximum number of NDP in the DGNs from the source node s is equal to the number of its neighbors (i.e. the node degree). Accordingly, the maximum number of destination nodes is equal to four. Since the DGN is vertex symmetric, we assume the source node is $s = (0,0)$.

For a particular set of destination nodes T, there are more than one possible NDP from s to T. For example, consider the network G_3 in Figure 3.2, let the source node be $s = (0,0)$ and the set of destination nodes be $T = \{(1,2),(-2,1),(-1,-1),(1,-1)\}$. Then, two NDP are given in Figure 3.3. In our work we give one of these NDP and it is denoted by $\mathbb{P}(s,T)$.

In this chapter, we denote the sum of the lengths of the shortest distances by

$$L(T) = \sum_{j=1}^{4} W(t_j).$$

For example in Figure 3.3a, $L(T) = 3+3+2+2 = 10$. Also, we denote the sum of the lengths of the NDP in $\mathbb{P}(s,T)$ by

$$|\mathbb{P}(s,T)| = \sum_{j=1}^{4} |P(s,t_j)|.$$

For example in Figure 3.3a, $|\mathbb{P}(s,T)| = 3+3+2+2 = 10$ which means the NDP in $\mathbb{P}(s,T)$ are the shortest paths because $|\mathbb{P}(s,T)| = L(T)$. In Figure 3.3b, $|\mathbb{P}(s,T)| = 3+3+4+2 = 12$. Since we assume that $s = (0,0)$, $|\mathbb{P}(s,T)|$ can be expressed in terms of the sum of the extra hops by

$$|\mathbb{P}(s,T)| = L(T) + \sum_{j=1}^{4} \delta(t_j).$$

We use this expression to compare between the sum of the lengths of the NDP given by the proposed algorithm and the sum of the lengths of the shortest paths throughout this chapter.

The following section describes our routing algorithm from the source node s to each of the four destination nodes in T using NDP.

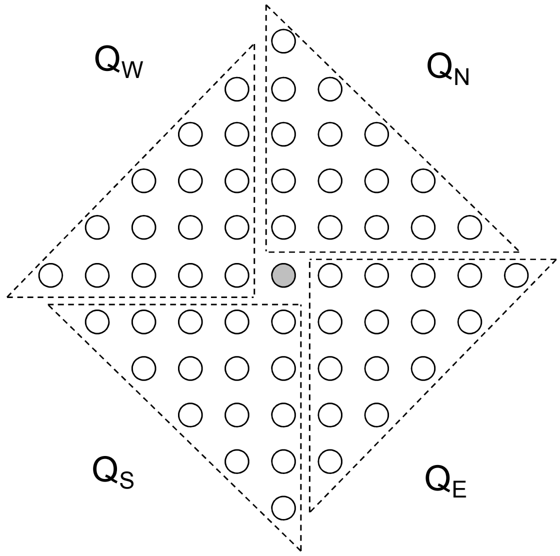


Figure 3.4: Quadrants in G_5

3.2 One-to-Many Node Disjoint Paths Routing

The basic idea of our routing algorithm is to design a set of distinctive and comprehensive cases based on the destination nodes' locations in the dense Gaussian networks (DGNs), and then construct the one-to-many node disjoint paths (NDP) $\mathbb{P}(s, T)$ for each case. The algorithm (see Algorithm 3) consists of two steps: case determination and NDP construction.

3.2.1 Step 1: Case Determination

Any DGN G_k can be partitioned into four non-overlapped quadrants based on the source node's address. For any source node $s = (s_x, s_y)$, these quadrants are:

1. $Q_N = \{(x, y) \in G_k \mid (x \geq s_x) \wedge (y \geq s_y + 1)\}$ (The north quadrant)

2. $Q_W = \{(x, y) \in G_k \mid (x \leq s_x - 1) \wedge (y \geq s_y)\}$ (The west quadrant)

Algorithm 3 One-to-Many NDP Routing in the Dense Gaussian Network G_k

Input: G_k, $T = \{t_j = (t_{j_x}, t_{j_y}) | 1 \leq j \leq 4\}$, $s = (s_x, s_y) \notin T$
Output: $\mathbb{P}(s, T)$
procedure $OneToMany\_NDP(G_k, T, s)$
  $|Q_N| = |Q_W| = |Q_S| = |Q_E| = 0;$
  for $1 \leq j \leq 4$ **do** ▷ Step 1
    if $(t_{j_x} \geq s_x) \wedge (t_{j_y} \geq s_y + 1)$ **then**
      $|Q_N| = |Q_N| + 1;$
    else if $(t_{j_x} \leq s_x - 1) \wedge (t_{j_y} \geq s_y)$ **then**
      $|Q_W| = |Q_W| + 1;$
    else if $(t_{j_x} \leq s_x) \wedge (t_{j_y} \leq s_y - 1)$ **then**
      $|Q_S| = |Q_S| + 1;$
    else
      $|Q_E| = |Q_E| + 1;$
    end if
  end for
  switch $\langle |Q_N|, |Q_W|, |Q_S|, |Q_E| \rangle$ ▷ Step 2
    $\langle 1,1,1,1 \rangle : \mathbb{P}(s,T) = Case1(G_k, T, s);$
    $\langle 2,0,2,0 \rangle \vee \langle 0,2,0,2 \rangle : \mathbb{P}(s,T) = Case2(G_k, T, s);$
    $\langle 2,2,0,0 \rangle \vee \langle 0,2,2,0 \rangle \vee \langle 0,0,2,2 \rangle \vee \langle 2,0,0,2 \rangle : \mathbb{P}(s,T) = Case3(G_k, T, s);$
    $\langle 2,1,1,0 \rangle \vee \langle 0,2,1,1 \rangle \vee \langle 1,0,2,1 \rangle \vee \langle 1,1,0,2 \rangle : \mathbb{P}(s,T) = Case4(G_k, T, s);$
    $\langle 2,0,1,1 \rangle \vee \langle 1,2,0,1 \rangle \vee \langle 1,1,2,0 \rangle \vee \langle 0,1,1,2 \rangle : \mathbb{P}(s,T) = Case5(G_k, T, s);$
    $\langle 2,1,0,1 \rangle \vee \langle 1,2,1,0 \rangle \vee \langle 0,1,2,1 \rangle \vee \langle 1,0,1,2 \rangle : \mathbb{P}(s,T) = Case6(G_k, T, s);$
    $\langle 3,0,0,1 \rangle \vee \langle 1,3,0,0 \rangle \vee \langle 0,1,3,0 \rangle \vee \langle 0,0,1,3 \rangle : \mathbb{P}(s,T) = Case7(G_k, T, s);$
    $\langle 3,1,0,0 \rangle \vee \langle 0,3,1,0 \rangle \vee \langle 0,0,3,1 \rangle \vee \langle 1,0,0,3 \rangle : \mathbb{P}(s,T) = Case8(G_k, T, s);$
    $\langle 3,0,1,0 \rangle \vee \langle 0,3,0,1 \rangle \vee \langle 1,0,3,0 \rangle \vee \langle 0,1,0,3 \rangle : \mathbb{P}(s,T) = Case9(G_k, T, s);$
    $\langle 4,0,0,0 \rangle \vee \langle 0,4,0,0 \rangle \vee \langle 0,0,4,0 \rangle \vee \langle 0,0,0,4 \rangle : \mathbb{P}(s,T) = Case10(G_k, T, s);$
  end switch
  return $\mathbb{P}(s,T);$
end procedure

3. $Q_S = \{(x,y) \in G_k \mid (x \leq s_x) \wedge (y \leq s_y - 1)\}$ (The south quadrant)

4. $Q_E = \{(x,y) \in G_k \mid (x \geq s_x + 1) \wedge (y \leq s_y)\}$ (The east quadrant)

Each quadrant has exactly $k(k+1)/2$ nodes where k is the network diameter. Figure 3.4 shows an example where $s = (0,0)$. In this example, the number of nodes

Table 3.1: All cases of $\langle |Q_N|, |Q_W|, |Q_S|, |Q_E| \rangle$

Case No.	Chosen Cases	Equivalent Cases
1	$\langle 1,1,1,1 \rangle$	no equivalent case
2	$\langle 2,0,2,0 \rangle$	$\langle 0,2,0,2 \rangle$
3	$\langle 2,2,0,0 \rangle$	$\langle 0,2,2,0 \rangle$, $\langle 0,0,2,2 \rangle$, $\langle 2,0,0,2 \rangle$
4	$\langle 2,1,1,0 \rangle$	$\langle 0,2,1,1 \rangle$, $\langle 1,0,2,1 \rangle$, $\langle 1,1,0,2 \rangle$
5	$\langle 2,0,1,1 \rangle$	$\langle 1,2,0,1 \rangle$, $\langle 1,1,2,0 \rangle$, $\langle 0,1,1,2 \rangle$
6	$\langle 2,1,0,1 \rangle$	$\langle 1,2,1,0 \rangle$, $\langle 0,1,2,1 \rangle$, $\langle 1,0,1,2 \rangle$
7	$\langle 3,0,0,1 \rangle$	$\langle 1,3,0,0 \rangle$, $\langle 0,1,3,0 \rangle$, $\langle 0,0,1,3 \rangle$
8	$\langle 3,1,0,0 \rangle$	$\langle 0,3,1,0 \rangle$, $\langle 0,0,3,1 \rangle$, $\langle 1,0,0,3 \rangle$
9	$\langle 3,0,1,0 \rangle$	$\langle 0,3,0,1 \rangle$, $\langle 1,0,3,0 \rangle$, $\langle 0,1,0,3 \rangle$
10	$\langle 4,0,0,0 \rangle$	$\langle 0,4,0,0 \rangle$, $\langle 0,0,4,0 \rangle$, $\langle 0,0,0,4 \rangle$

in each quadrant is equal to $5(5+1)/2 = 15$ where the diameter $k = 5$.

Based on this network partitioning, the algorithm determines the current case. Let $|Q_i| \in \{0,1,2,3,4\}$ be the number of destination nodes in the quadrant Q_i for $i = N, W, S, E$. Let the ordered set $\langle |Q_N|, |Q_W|, |Q_S|, |Q_E| \rangle$ represent the number of destination nodes in each quadrant such that $|Q_N| + |Q_W| + |Q_S| + |Q_E| = 4$. For example, $\langle 4,0,0,0 \rangle$ means all destination nodes are in the north quadrant.

Since there are four destination nodes that are distributed over the four quadrants, there are exactly $\binom{4+4-1}{4} = 35$ possibilities of $\langle |Q_N|, |Q_W|, |Q_S|, |Q_E| \rangle$. The one-to-many NDP routing algorithm must construct all NDP $\mathbb{P}(s,T)$ for each one of these 35 possibilities. However, since G_k is vertex symmetric, the solution for $\langle x_1, x_2, x_3, x_4 \rangle$ is equivalent to the solutions for $\langle x_4, x_1, x_2, x_3 \rangle$, $\langle x_3, x_4, x_1, x_2 \rangle$, and $\langle x_2, x_3, x_4, x_1 \rangle$ (by rotation[1]) where $x_1, x_2, x_3, x_4 \in \{0,1,2,3,4\}$ and $\sum_{i=1}^{4} x_i = 4$. So in this work, we show the NDP $\mathbb{P}(s,T)$ for 10 cases. The solutions for these 10

[1]Multiplying all nodes by $\mathbf{i}$ rotates the network in the counterclockwise direction [31].

cases are equivalent to the solutions for all 35 cases. Table 3.1 shows the chosen 10 cases and the equivalent cases.

Based on the destination nodes' addresses, the algorithm evaluates $\langle |Q_N|, |Q_W|, |Q_S|, |Q_E| \rangle$ in the first step. In the second step, the algorithm constructs the NDP $\mathbb{P}(s, T)$.

3.2.2 Step 2: One-to-Many NDP Construction

In this step, the algorithm constructs four NDP from the source node to the destination nodes based on the case determined during the first step. In the following, we describe the NDP construction for each of these 10 cases. Before that, we need the following definitions.

Definition 3.2.1. *In a DGN G_k where k is the diameter, let the source node be $(0,0)$. Then,* the north, west, south, and east paths *start with* $(0,0) \to (0,1)$*,* $(0,0) \to (-1,0)$*,* $(0,0) \to (0,-1)$*, and* $(0,0) \to (1,0)$ *respectively.*

Definition 3.2.2. *In a DGN G_k where k is the diameter, let $t_j = (t_{j_x}, t_{j_y}) \in Q_i$ for $j = 1,2,3,4$ and $i = N, W, S, E$ be any destination node. Then, the destination node t_j is:*

- *the* top *destination node of Q_i if $t_{j_y} = \max\{t_{r_y} | t_r = (t_{r_x}, t_{r_y}) \in Q_i\}$,*
- *the* bottom *destination node of Q_i if $t_{j_y} = \min\{t_{r_y} | t_r = (t_{r_x}, t_{r_y}) \in Q_i\}$,*
- *the* left *destination node of Q_i if $t_{j_x} = \min\{t_{r_x} | t_r = (t_{r_x}, t_{r_y}) \in Q_i\}$,*
- *the* right *destination node of Q_i if $t_{j_x} = \max\{t_{r_x} | t_r = (t_{r_x}, t_{r_y}) \in Q_i\}$,*

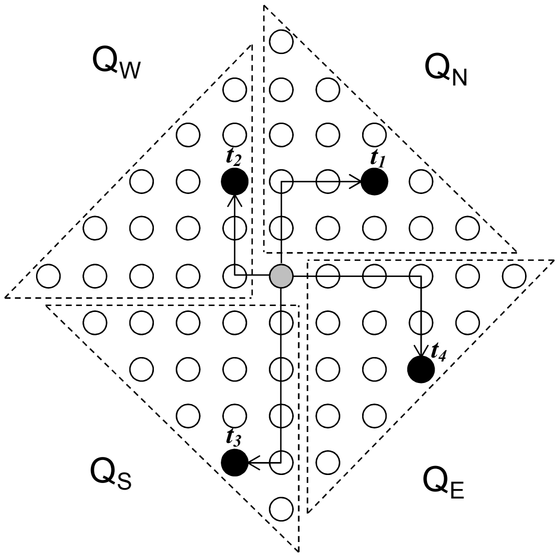


Figure 3.5: Example of Case 1 (G_5)

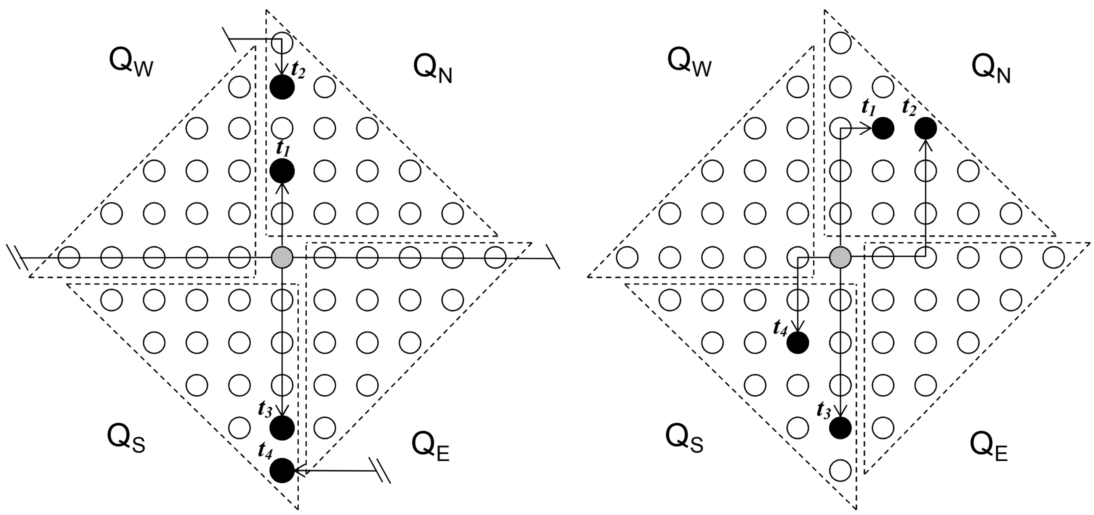
- *the* max-weight *destination node of* Q_i *if* $W(t_j) = \max\{W(t_r)|t_r = (t_{r_x}, t_{r_y}) \in Q_i\}$, *and/or*

- *the* min-weight *destination node of* Q_i *if* $W(t_j) = \min\{W(t_r)|t_r = (t_{r_x}, t_{r_y}) \in Q_i\}$.

Note that the top, bottom, left, right, max-weight, or min-weight destination node as defined in Definition 3.2.2 is not necessarily unique. So, we say, for example, top/left of Q_i to uniquely specify a destination node in case the top destination node is not unique by choosing the most left destination node among those top destination nodes.

Now, we explain how to construct the NDP for each case.

Case 1 $\langle 1, 1, 1, 1\rangle$



(a) Case 2.1 (b) Case 2.2

Figure 3.6: Examples of Case 2 (G_5)

In this case, each quadrant has exactly one destination node; this is the most simple case.

Lemma 3.2.1. *Let the case be* $\langle 1, 1, 1, 1\rangle$*. Then, there exist NDP* $\mathbb{P}(s, T)$ *such that* $|\mathbb{P}(s, T)| = L(T)$*.*

Proof. The NDP to the destination nodes in the north, west, south, and east quadrants are connected along the north, west, south, and east paths respectively. (The paths are straight forward and can be immediately gleaned from the example shown in Figure 3.5.) Clearly, $|\mathbb{P}(s, T)| = L(T)$. □

Case 2 $\langle 2, 0, 2, 0\rangle$

In this case, the north quadrant Q_N and the south quadrant Q_S have two destination nodes each.

Lemma 3.2.2. *Let the case be* $\langle 2, 0, 2, 0\rangle$*. Then, there exist NDP* $\mathbb{P}(s, T)$ *such that* $L(T) \leq |\mathbb{P}(s, T)| \leq L(T) + (4k - 6)$*.*

Proof. Let $t_1, t_2 \in Q_N$ and $t_3, t_4 \in Q_S$. The NDP to t_1 and t_2 are connected along the north and east paths; and the NDP to t_3 and t_4 are connected along the west and south paths. Here, we show only the NDP to t_1 and t_2. The NDP to t_3 and t_4 can be similarly constructed. All paths can be gleaned from Figure 3.6.

Depending on the addresses of t_1 and t_2, we have the following subcases:

Case 2.1 $\boldsymbol{t_{1_x} = t_{2_x} = 0}$**:** In this case, we reach the bottom and top destination nodes using the north and east paths respectively.(Figure 3.6a shows an example.)

Let t_1 and t_2 be the bottom and top destination nodes of Q_N respectively. Then, the north path is $P(s, t_1)$ as $(0, 0) \to (0, 1) \to \cdots \to (0, t_{1_y})$. The east path is $P(s, t_2)$ as $(0, 0) \to (1, 0) \to \cdots \to (k, 0) \to (k, 0)^E = (0, k) \to (0, k - 1) \to \cdots \to (0, t_{2_y})$.

The length of $P(s, t_1)$ equals $W(t_1)$. For $P(s, t_2)$, the minimum and maximum lengths of this path occur respectively when $t_2 = (0, k)$ and $t_2 = (0, 2)$. It follows that the length of $P(s, t_2)$ is between $W((0, k)) + \delta((0, k)) = W((0, k)) + 1$ and $W((0, 2)) + \delta((0, 2)) = W((0, 2)) + (2k - 3)$.

Case 2.2 $\boldsymbol{t_{1_x} \neq 0}$ **or** $\boldsymbol{t_{2_x} \neq 0}$**:**

In this case, there exists at least one destination node that its x value is not equal to zero. We reach this destination node using the east path and the other destination node using the north path.(Figure 3.6b shows an example.)

If there is a destination node that its $x = 0$, let it be t_1. If there is no such

Table 3.2: All subcases of Case 2

Case	Lower Bound	Upper Bound
$t_{1_x} = t_{2_x} = t_{3_x} = t_{4_x} = 0$	$L(T) + 2$	$L(T) + (4k - 6)$
$(t_{1_x} \neq 0$ or $t_{2_x} \neq 0)$and$(t_{3_x} \neq 0$ or $t_{4_x} \neq 0)$	$L(T)$	$L(T)$
$(t_{1_x} = t_{2_x} = 0)$and$(t_{3_x} \neq 0$ or $t_{4_x} \neq 0)$	$L(T) + 1$	$L(T) + (2k - 3)$
$(t_{1_x} \neq 0$ or $t_{2_x} \neq 0)$and$(t_{3_x} = t_{4_x} = 0)$	$L(T) + 1$	$L(T) + (2k - 3)$

a destination node, let the top/left destination node of Q_N be t_1. In either cases, let the other destination node be t_2. Then, the north path is $P(s, t_1)$ and the east path is $P(s, t_2)$. (Both paths are straight forward and can be immediately gleaned from the example shown in Figure 3.6b.) Clearly, $|P(s, t_1)| = W(t_1)$ and $|P(s, t_2)| = W(t_2)$.

Similarly, we can construct all NDP for all possibilities of Case 2. Table 3.2 shows the upper and lower bounds of these possibilities. It follows that for Case 2, $L(T) \leq |\mathbb{P}(s, T)| \leq L(T) + (4k - 6)$. □

In the upcoming cases, Case 2 is used to reach two destination nodes in Q_N using the east and north paths as long as none of the following nodes is used: $(1, 0), (2, 0), \ldots, (k-1, 0), (k, 0)$. Similarly, Case 2 is used to reach two destination nodes in Q_S using the west and south paths as long as none of the following nodes is used: $(-1, 0), (-2, 0), \ldots, (-k+1, 0), (-k, 0)$.

Case 3: $\langle 2, 2, 0, 0 \rangle$

In this case, the north and west quadrants have two destination nodes each.

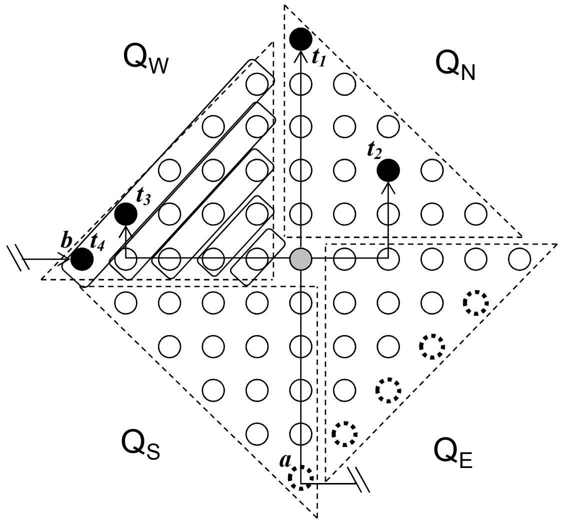


Figure 3.7: Example of Case 3 (G_5)

Lemma 3.2.3. *Let the case be* $\langle 2,2,0,0\rangle$. *Then, there exist NDP* $\mathbb{P}(s,T)$ *such that* $L(T)+1 \le |\mathbb{P}(s,T)| \le L(T)+(4k-6)$.

Proof. Let $t_1, t_2 \in Q_N$ and $t_3, t_4 \in Q_W$. The path to the min-weight/right destination node (say t_3) of Q_W is connected along the west path $P(s,t_3)$; and the path to the max-weight/left destination node (i.e. t_4) of Q_W is connected along the south path $P(s,t_4)$. (These paths can be immediately gleaned from the example shown in Figure 3.7 where $b = (b_x, t_{4_y}) \in Q_W$ is a border node and $a = (a_x, a_y) = b^W \in Q_E$.) Figure 3.7 shows all possibilities (dashed nodes) of node a. Clearly there is no node of $(1,0),(2,0),\ldots,(k-1,0),(k,0)$ is used. That means we can safely apply Case 2 to reach t_1 and t_2 using the north and east paths. It follows that $(W(t_1)+W(t_2)) \le (|P(s,t_1)|+|P(s,t_2)|) \le (W(t_1)+W(t_2)+2k-3)$. The length of $P(s,t_3)$ is equal to $W(t_3)$. The length of $P(s,t_4)$ is at most $W(t_4)+(2k-3)$ and this occurs when $W(t_4)=2$. Also, this length is at least $W(t_4)+1$ and this occurs when $W(t_4)=k$. It follows that $L(T)+1 \le |\mathbb{P}(s,T)| \le L(T)+(4k-6)$. $\square$

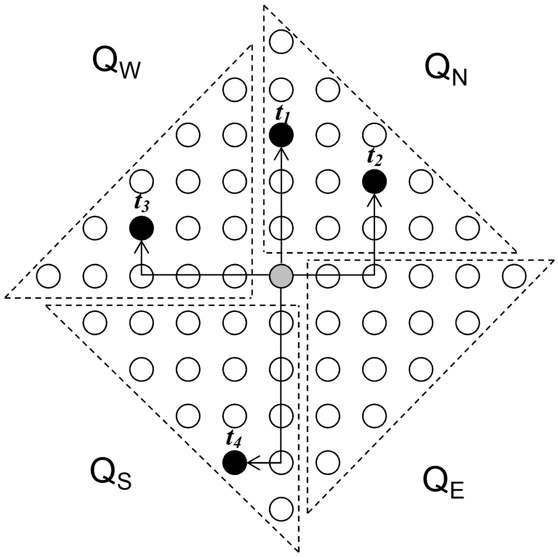


Figure 3.8: Example of Case 4 (G_5)

Case 4: $\langle 2, 1, 1, 0 \rangle$

In this case, there exist two destination nodes in Q_N, one destination node in Q_W, and one destination node in Q_S.

Lemma 3.2.4. *Let the case be $\langle 2, 1, 1, 0 \rangle$. Then, there exist NDP $\mathbb{P}(s, T)$ such that $L(T) \leq |\mathbb{P}(s, T)| \leq L(T) + (2k - 3)$.*

Proof. Let $t_1, t_2 \in Q_N$, $t_3 \in Q_W$, and $t_4 \in Q_S$. $P(s, t_1)$ and $P(s, t_2)$ are obtained by applying Case 2. It follows that $(W(t_1) + W(t_2)) \leq (|P(s, t_1)| + |P(s, t_2)|) \leq (W(t_1) + W(t_2) + 2k - 3)$. ($P(s, t_3)$ and $P(s, t_4)$ can be immediately gleaned from the example shown in Figure 3.8.) The sum of their lengths equals $W(t_3) + W(t_4)$. It follows that $L(T) \leq |\mathbb{P}(s, T)| \leq L(T) + (2k - 3)$. $\square$

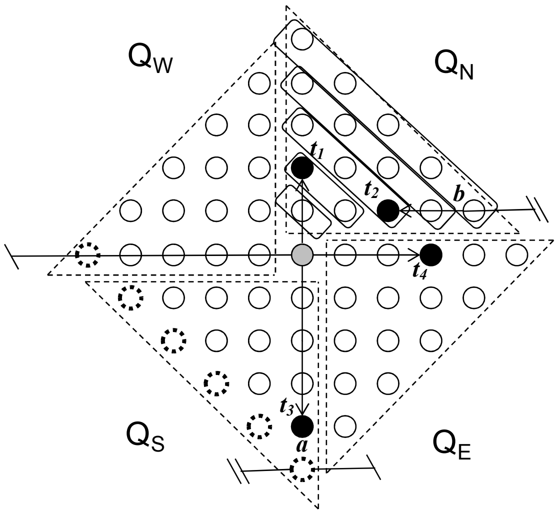


Figure 3.9: Example of Case 5 (G_5)

Case 5: $\langle 2, 0, 1, 1 \rangle$

In this case, there exist two destination nodes in Q_N, one destination node in Q_S, and one destination node in Q_E.

Lemma 3.2.5. *Let the case be $\langle 2, 0, 1, 1 \rangle$. Then, there exist NDP $\mathbb{P}(s, T)$ such that $L(T) + 1 \leq |\mathbb{P}(s, T)| \leq L(T) + (2k - 2)$.*

Proof. Let $t_1, t_2 \in Q_N$, $t_3 \in Q_S$, and $t_4 \in Q_E$. The following steps construct the NDP:

1. Reach the min-weight/left destination node of Q_N (Say t_1) and t_4 using the north and east paths. (Figure 3.9 shows as example for $P(s, t_1)$ and $P(s, t_4)$.) The sum of their lengths equals $W(t_1) + W(t_4)$.

2. Connect the other destination node in Q_N (i.e. t_2) with the border node $b = (b_x, t_{2_y}) \in Q_N$ horizontally using the path $P(b, t_2)$ as $(b_x, t_{2_y}) \rightarrow (b_x -$

$1, t_{2_y}) \to \cdots \to (t_{2_x}, t_{2_y})$.

3. Let $a = (a_x, a_y)$ be either the east (b^E) or north (b^N) neighbor of node b depending on the location of t_3 in Q_S as follows (the dashed nodes in Figure 3.9 represent all possibilities of node a):

$$a = \begin{cases} b^N & \text{if} \quad b^E = t_3 \\ b^E & \text{if} \quad b^E \neq t_3 \end{cases}$$

   Note that node a can be either in the west quadrant Q_W (if $t_2 = (0, k)$ and $t_3 = (-(k-1), -1)$) or the south quadrant Q_S (otherwise).

   This step is always possible because by the network connectivity each border node b in Q_N is connected with two nodes (b^N and b^E) using the wraparound links. One of these two nodes must be available to use because there exists only one destination node in Q_S. In case a is in Q_W, step four is still valid because Q_W has no destination node.

4. If node a is in the west quadrant Q_W, the west path $P(s, a)$ and the south path $P(s, t_3)$ are exactly same as the west and south paths in Case 1 respectively. The sum of their lengths equals $W(a) + W(t_4) = k + W(t_4)$. If node a is in the south quadrant Q_S, $P(s, a)$ and $P(s, t_3)$ are obtained by applying Case 2. It follows that $(k+W(t_3)) \leq (|P(s,a)|+|P(s,t_3)|) \leq (W(t_3)+3k-3)$.

   The length of $P(s, t_2)$ as $P(s, a) \to P(b, t_2)$ is at most $W(t_2) + (2k-2)$ and this occurs when $W(t_2) = 2$ and $a = (0, -k)$. Also, this length is at least $W(t_2) + 1$ and this occurs when $W(t_2) = k$ and the following is not true: $a_x = t_{3_x} = 0$.

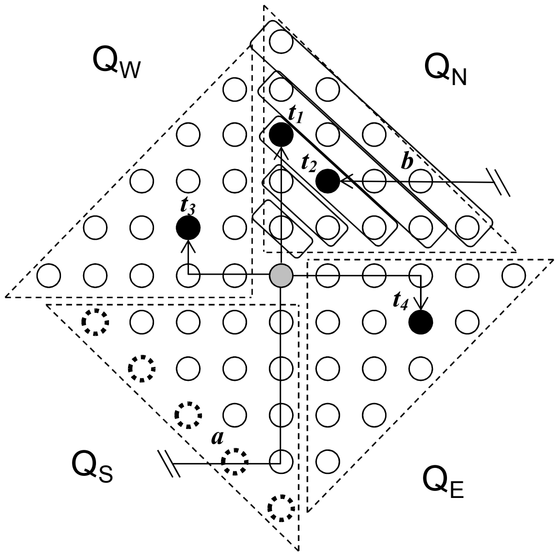


Figure 3.10: Example of Case 6 (G_5)

It follows that $L(T) + 1 \leq |\mathbb{P}(s, T)| \leq L(T) + (2k - 2)$. $\square$

Case 6: $\langle 2, 1, 0, 1 \rangle$

In this case, there exist two destination nodes in Q_N, one destination node in Q_W, and one destination node in Q_E.

Lemma 3.2.6. *Let the case be $\langle 2, 1, 0, 1 \rangle$. Then, there exist NDP $\mathbb{P}(s, T)$ such that $L(T) + 1 \leq |\mathbb{P}(s, T)| \leq L(T) + (2k - 3)$.*

Proof. Let $t_1, t_2 \in Q_N$, $t_3 \in Q_W$, and $t_4 \in Q_E$. Then, the path to the min-weight/left destination node (say t_1) of Q_N is connected along the north path $P(s, t_1)$; and the path to the max-weight/right destination node (i.e. t_2) of Q_N is connected along the south path $P(s, t_2)$. The paths to t_3 and t_4 are connected along the east and west paths respectively. (All paths can be immediately gleaned

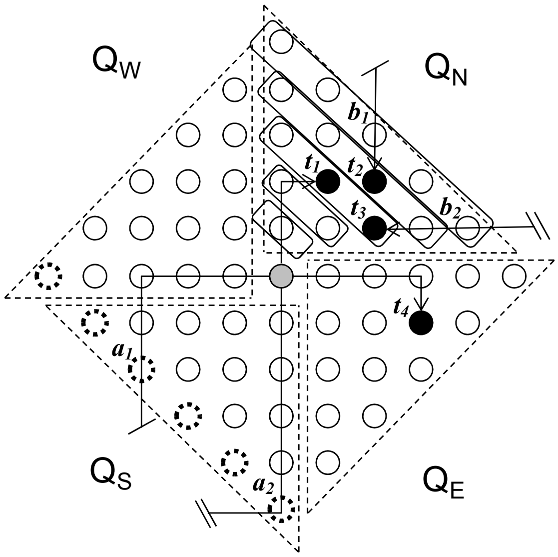


Figure 3.11: Example of Case 7 (G_5)

from the example shown in Figure 3.10 where $b = (b_x, t_{2_y}) \in Q_N$ is a border node and $a = (a_x, a_y) = b^E \in Q_S$.)

The sum of lengths of $P(s,t_1)$, $P(s,t_3)$, and $P(s,t_4)$ is equal to $W(t_1)+W(t_3)+W(t_4)$. The length of $P(s,t_2)$ is at most $W(t_2) + (2k-3)$ and this occurs when $W(t_2) = 2$. Also, this length is at least $W(t_2)+1$ and this occurs when $W(t_2) = k$. It follows that $L(T) + 1 \leq |\mathbb{P}(s,T)| \leq L(T) + (2k-3)$. □

Case 7: $\langle 3, 0, 0, 1 \rangle$

In this case, there exist three destination nodes in Q_N and one destination node in Q_E.

Lemma 3.2.7. *Let the case be $\langle 3, 0, 0, 1 \rangle$. Then, there exist NDP $\mathbb{P}(s,T)$ such that $L(T) + 2 \leq |\mathbb{P}(s,T)| \leq L(T) + (4k-6)$.*

Proof. Let $t_1, t_2, t_3 \in Q_N$, and $t_4 \in Q_E$. The following steps construct the NDP (see Figure 3.11):

1. Reach the min-weight/left destination node of Q_N (say t_1) and t_4 using the north and east paths. (Figure 3.9 shows an example of $P(s, t_1)$ and $P(s, t_4)$.) The sum of their lengths equals $W(t_1) + W(t_4)$.

2. After the previous step, the remaining destination nodes in Q_N are t_2 and t_3. Among these two destination nodes, connect the top/left destination node of Q_N (say t_2), with the border node $b_1 = (t_{2_x}, b_{1_y}) \in Q_N$ vertically using the path $P(b_1, t_2)$ as $(t_{2_x}, b_{1_y}) \to (t_{2_x}, b_{1_y} - 1) \to \cdots \to (t_{2_x}, t_{2_y})$. Also, connect the last destination node in Q_N (i.e t_3) with the border node $b_2 = (b_{2_x}, t_{3_y}) \in Q_N$ horizontally using the path $P(b_2, t_3)$ as $(b_{2_x}, t_{3_y}) \to (b_{2_x} - 1, t_{3_y}) \to \cdots \to (t_{3_x}, t_{3_y})$.

   Constructing the path $P(b_1, t_2)$ vertically and the path $P(b_2, t_3)$ horizontally is important to maintain the disjointness condition for two reasons: 1) in Q_N, $P(b_1, t_2)$ and $P(b_2, t_3)$ are always node disjoint regardless of the locations of t_2 and t_3, and 2) in Q_S and Q_W, the north neighbor of b_1 and the east neighbor of b_2 are always different nodes.

3. Let the north neighbor of b_1 be $a_1 = b_1^N$ and the east neighbor of b_2 be $a_2 = b_2^E$. Figure 3.11 shows all possibilities of a_1 and a_2 (the dashed nodes). Note that it is possible that a_2 exists in Q_W. So, apply Case 5 (not Case 2) to connect the source node with a_1 and a_2.

   The length of $P(s, t_2)$ as $P(s, a_1) \to P(b_1, t_2)$ is at most $W(t_2) + (2k - 3)$ and this occurs when $W(t_2) = 2$. Also, this length is at least $W(t_2) + 1$ and this

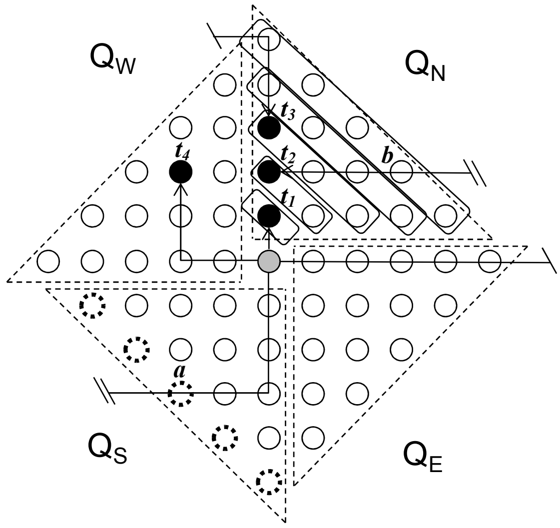


Figure 3.12: Example of Case 8 (G_5)

occurs when $W(t_2) = k$. The length of $P(s, t_2)$ as $P(s, a_1) \to P(b_1, t_2)$ is at most $W(t_2) + (2k - 3)$ and this occurs when $W(t_2) = 2$. Also, this length is at least $W(t_2) + 1$ and this occurs when $W(t_2) = k$.

It follows that $L(T) + 2 \leq |\mathbb{P}(s, T)| \leq L(T) + (4k - 6)$. $\square$

Case 8: $\langle 3, 1, 0, 0 \rangle$

In this case, there exist three destination nodes in Q_N and one destination node in Q_W.

Lemma 3.2.8. *Let the case be $\langle 3, 1, 0, 0 \rangle$. Then, there exist NDP $\mathbb{P}(s, T)$ such that $L(T) + 1 \leq |\mathbb{P}(s, T)| \leq L(T) + (4k - 6)$.*

Proof. Let $t_1, t_2, t_3 \in Q_N$, and $t_4 \in Q_W$. Let the destination node t_2 be $t_{1_y} < t_{2_y} < t_{3_y}$ if $t_{1_x} = t_{2_x} = t_{3_x} = 0$. Otherwise, let the destination node t_2 be the max-weight/right destination node of Q_N. Then, t_2 and t_4 are connected along the south and west paths respectively. ($P(s,t_2)$ and $P(s,t_4)$ can be immediately gleaned from the example shown in Figure 3.12 where $b = (b_x, t_{2_y}) \in Q_W$ is a border node and $a = (a_x, a_y) = b^E \in Q_S$ is the *east* neighbor of b.) Figure 3.12 also shows all possibilities (dashed nodes) of node a. Using the east neighbor of b is important because in this way all possibilities of a are in Q_S which has no destination node. It follows that, the length of $P(s,t_2)$ equals $W(t_2)+1 \leq |P(s,t_2)| \leq W(t_2)+(2k-3)$ and the length of $P(s,t_4)$ equals $W(t_4)$.

The paths to the remaining destination nodes in Q_1 (i.e. t_1 and t_3) are obtained by applying Case 2. The sum of their lengths is $W(t_1) + W(t_3) \leq |P(s,t_1)| + |P(s,t_3)| \leq W(t_1) + W(t_3) + (2k-3)$. It follows that $L(T) + 1 \leq |\mathbb{P}(s,T)| \leq L(T) + (4k-6)$. $\square$

Case 9: $\langle 3, 0, 1, 0 \rangle$

In this case, there exist three destination nodes in Q_N and one destination node in Q_S.

Lemma 3.2.9. *Let the case be* $\langle 3, 0, 1, 0 \rangle$*. Then, there exist NDP* $\mathbb{P}(s,T)$ *such that* $L(T) + 1 \leq |\mathbb{P}(s,T)| \leq L(T) + (4k-5)$*.*

Proof. Let $t_1, t_2, t_3 \in Q_N$, and $t_4 \in Q_S$. Let the destination node t_2 be $t_{1_y} < t_{2_y} < t_{3_y}$ if $t_{1_x} = t_{2_x} = t_{3_x} = 0$. Otherwise, let the destination node t_2 be the max-weight/right destination node of Q_N. Then, t_2 and t_4 are connected along the south and west paths respectively. The process of constructing $P(s,t_2)$ and $P(s,t_4)$ is exactly same as the process of constructing $P(s,t_2)$ and $P(s,t_4)$ in Case 5. It

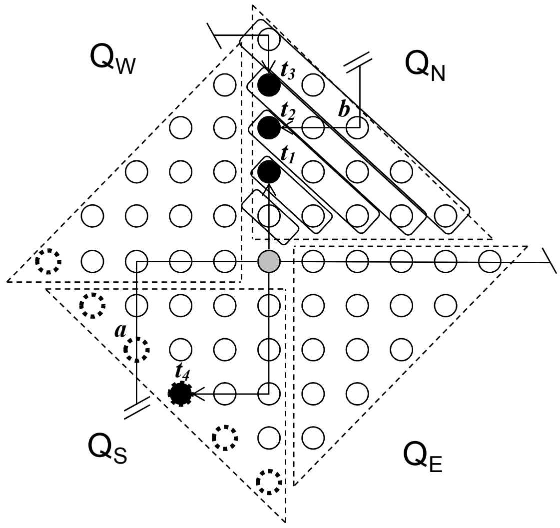


Figure 3.13: Example of Case 9 (G_5)

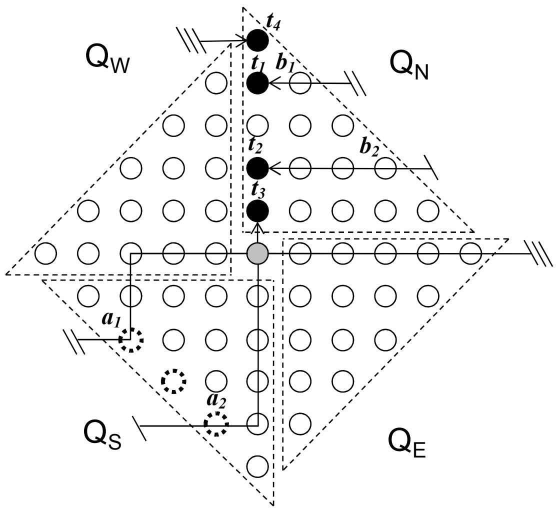
follows that the sum of the lengths of $P(s,t_2)$ and $P(s,t_4)$ is $W(t_2)+W(t_4)+1 \leq |P(s,t_2)| + |P(s,t_4)| \leq W(t_2) + W(t_4) + (2k-2)$. The paths to the remaining destination nodes in Q_1 (i.e. t_1 and t_3) are obtained by applying Case 2. The sum of their lengths is $W(t_1)+W(t_3) \leq |P(s,t_1)|+|P(s,t_3)| \leq W(t_1)+W(t_3)+(2k-3)$. It follows that $L(T)+1 \leq |\mathbb{P}(s,T)| \leq L(T)+(4k-5)$. □

Case 10: $\langle 4,0,0,0 \rangle$

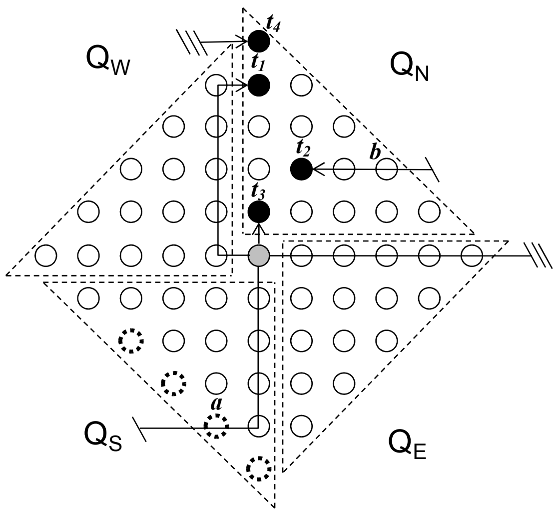
In this case, all four destination nodes are in Q_N; and this is the most sophisticated case.

Lemma 3.2.10. *Let the case be* $\langle 4,0,0,0 \rangle$*. Then, there exist NDP* $\mathbb{P}(s,T)$ *such that* $L(T)+2 \leq |\mathbb{P}(s,T)| \leq L(T)+(6k-11)$.

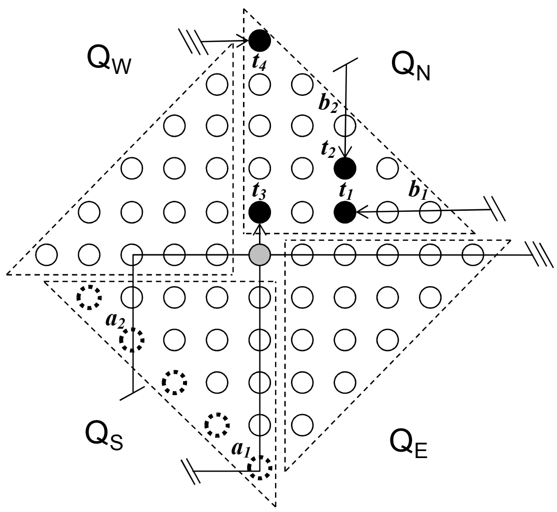
Proof. Here, $t_1,t_2,t_3,t_4 \in Q_N$. To show the construction of the NDP precisely, we



(a) Case 10.1



(b) Case 10.2



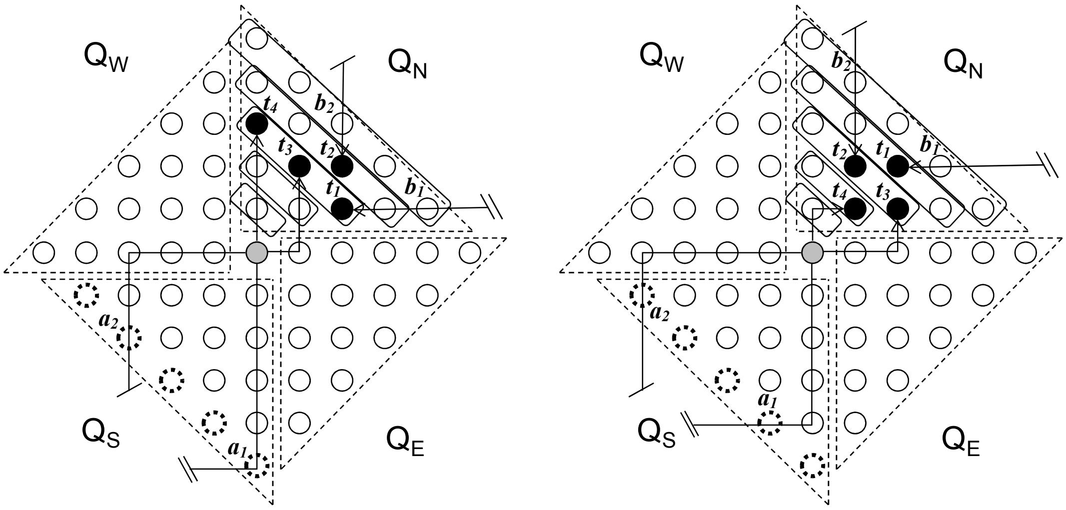
(c) Case 10.3

Figure 3.14: Examples of Case 10 (G_5)

divide this case into the following subcases:

Case 10.1 Four destination nodes have $x = 0$:

In this case, $t_{1_x} = t_{2_x} = t_{3_x} = t_{4_x} = 0$. Let $t_{3_y} < t_{2_y} < t_{1_y} < t_{4_y}$. (The paths are straight forward and can be immediately gleaned from the example shown in Figure 3.14a where $b_1 = (b_{1_x}, t_{1_y}) \in Q_N$, $b_2 = (b_{2_x}, t_{2_y}) \in Q_N$,



(a) Case 10.4 (b) Case 10.5

Figure 3.15: Examples of Case 10 (G_5)

$a_1 = b_1^E \in Q_S$, and $a_2 = b_2^E \in Q_S$.)

The length of $P(s,t_1)$ is $W(t_1)+3 \leq P(s,t_1) \leq W(t_1)+(2k-5)$. The lower and upper bounds occur when $t_{1_y} = k-1$ and $t_{1_y} = 3$ respectively. The length of $P(s,t_2)$ is $W(t_2)+5 \leq P(s,t_2) \leq W(t_2)+(2k-3)$. The lower and upper bounds occur when $t_{2_y} = k-2$ and $t_{2_y} = 2$ respectively. The sum of lengths of $P(s,t_3)$ and $P(s,t_4)$ is $W(t_3)+W(t_4)+1 \leq |P(s,t_3)|+|P(s,t_3)| \leq W(t_3)+W(t_4)+(2k-7)$. The upper and lower bounds occur when $t_{4_y} = 4$ and $t_{4_y} = k$ respectively. It follows that $L(T)+9 \leq |\mathbb{P}(s,T)| \leq L(T)+(6k-15)$.

Case 10.2 Three destination nodes have $x = 0$:

Let $t_{1_x} = t_{3_x} = t_{4_x} = 0$, $t_{2_x} \neq 0$, and $t_{3_y} < t_{1_y} < t_{4_y}$. (The paths are straight forward and can be immediately gleaned from the example shown in Figure 3.14b where $b = (b_{1_x}, t_{2_y}) \in Q_N$ and $a = b^E \in Q_S$.)

The length of $P(s,t_2)$ is $W(t_2)+1 \leq P(s,t_2) \leq W(t_2)+(2k-3)$. The upper bound occurs when $W(t_2) = 2$. The length of $P(s, t_1)$ is equal to $W(t_1) + 2$. The sum of lengths of $P(s, t_3)$ and $P(s, t_4)$ is $W(t_3) + W(t_4) + 1 \leq |P(s, t_3)| + |P(s, t_3)| \leq W(t_3) + W(t_4) + (2k - 5)$. The upper and lower bounds occur when $t_{4_y} = 3$ and $t_{4_y} = k$ respectively. It follows that $L(T) + 4 \leq |\mathbb{P}(s, T)| \leq L(T) + (4k - 8)$.

Case 10.3 Two destination nodes have $x = 0$:

Let $t_{3_x} = t_{4_x} = 0$ and $t_{1_x}, t_{2_x} \neq 0$ (see Figure 3.14c). The following steps construct the NDP:

1. Apply Case 2.1 to reach t_3 and t_4 using the north and east paths. The sum of lengths of $P(s, t_3)$ and $P(s, t_4)$ is $W(t_3) + W(t_4) + 1 \leq |P(s, t_3)| + |P(s, t_3)| \leq W(t_3) + W(t_4) + (2k - 3)$. The upper and lower bounds occur when $t_{4_y} = 2$ and $t_{4_y} = k$ respectively.

2. Apply Case 7 to reach t_1 and t_2 using the west and south paths. The sum of lengths of $P(s, t_3)$ and $P(s, t_4)$ is $W(t_3) + W(t_4) + 2 \leq |P(s, t_3)| + |P(s, t_3)| \leq W(t_3) + W(t_4) + (4k - 8)$. The upper bound occurs when one of these destination nodes is node $(1, 1)$ and the weight of the other destination node is equal to three. The lower bound occurs when $W(t_3) = W(t_4) = k$.

It follows that $L(T) + 3 \leq |\mathbb{P}(s, T)| \leq L(T) + (6k - 11)$.

Case 10.4: One destination node has $x = 0$:

Let $t_{4_x} = 0$ and $t_{1_x}, t_{2_x}, t_{3_x} \neq 0$ (see Figure 3.15a). Also, let t_3 be the min-weight/left destination node among t_1,t_2, and t_3; and let t_1 and t_2 be

Table 3.3: Specifying the 1st and 2nd destination nodes in Case 10.5

No. of 1st min-weight	No. of 2nd min-weight	Choose
1	1	1st and 2nd min-weight
1	> 1	1st min-weight and right of 2nd min-weight
> 1	≥ 0	1st and 2nd right of 1st min-weight

respectively the top/left and bottom/right destination nodes only among t_1 and t_2. Then, the following steps construct the NDP:

1. Apply Case 2.2 to construct $P(s, t_3)$ and $P(s, t_4)$ using the north and east paths. The sum of lengths of $P(s, t_3)$ and $P(s, t_4)$ is equal to $W(t_3) + W(t_4)$.

2. Apply Case 7 to construct $P(s, t_1)$ and $P(s, t_2)$ using the south and west paths. The sum of lengths of $P(s, t_1)$ and $P(s, t_2)$ is $W(t_1)+W(t_2)+2 \leq |P(s, t_1)| + |P(s, t_2)| \leq W(t_1)+W(t_2)+(4k-10)$. The upper and lower bounds occur when $W(t_1) = W(t_2) = 3$ and $W(t_1) = W(t_2) = k$ respectively.

It follows that $L(T) + 2 \leq |\mathbb{P}(s, T)| \leq L(T) + (4k - 10)$.

Case 10.5 None of the destination nodes has $x = 0$:

In this case, $t_{1_x}, t_{2_x}, t_{3_x}, t_{4_x} \neq 0$ (see Figure 3.15b). The following steps construct the NDP:

1. Count the number of destination nodes whose weights are equal to the minimum weight among all destination nodes (1st min-weight). (For

Table 3.4: Specifying the 3rd and 4th destination nodes in Case 10.5

No. of 1st max-weight	No. of 2nd max-weight	Choose
1	1	1st and 2nd max-weight
1	> 1	1st max-weight and left of 2nd max-weight
> 1	≥ 0	1st and 2nd left of 1st max-weight

example in Figure 3.15b, the number of destination nodes in the 1st min-weight equals one (t_4).)

2. Count the number of destination nodes in the 2nd min-weight among all destination nodes. (For example in Figure 3.15b, the number of destination nodes in the 2nd min-weight equals two (t_2 and t_3).)

3. Use Table 3.3 to specify two destination nodes. Note that this table specifies exactly two destination nodes. Let these destination nodes be t_3 and t_4. (For the example given in Figure 3.15b, Table 3.3 specifies the destination node in the 1st min-weight (t_4) and the right destination node among those in the 2nd min-weight (t_3).)

4. Apply Case 2.2 to construct $P(s, t_3)$ and $P(s, t_4)$ using the north and east paths. The sum of lengths of $P(s, t_3)$ and $P(s, t_4)$ is equal to $W(t_3) + W(t_4)$.

5. Count the number of destination nodes which have the max-weight among all destination nodes (1st max-weight). (For example in Figure 3.15b, the number of destination nodes in the 1st max-weight equals one (t_1).)

6. Count the number of destination nodes in the 2nd max-weight among

all destination nodes. (For example in Figure 3.15b, the number of destination nodes in the 2^{nd} max-weight equals two (t_2 and t_3).)

7. Use Table 3.4 to specify two destination nodes. Note that these destination nodes are different from the destination nodes specified in Step 3. Let these destination nodes be t_1 and t_2. (For the example given in Figure 3.15b, Table 3.4 specifies the destination node in the 1^{st} max-weight (t_1) and the left destination node among those in the 2^{nd} max-weight (t_2).)

8. Apply Case 7 to construct $P(s, t_1)$ and $P(s, t_2)$ using the south and west paths with wraparound links. The sum of lengths of $P(s, t_1)$ and $P(s, t_2)$ is $W(t_1) + W(t_2) + 2 \leq |P(s, t_1)| + |P(s, t_2)| \leq W(t_1) + W(t_2) + (4k - 12)$. The upper bound occurs when the weight value of one of these destination nodes is equal to three and the weight value of the other destination nodes is equal to four. The lower bound occurs when $W(t_3) = W(t_4) = k$.

It follows that $L(T) + 2 \leq |\mathbb{P}(s, T)| \leq L(T) + (4k - 12)$.

After comparing the upper and lower bounds of all subcases of Case 10, the minimum lower bound and the maximum upper bound occur when the cases are Case 10.4 (or 10.5) and Case 10.3 respectively. It follows that for Case 10 $L(T) + 2 \leq |\mathbb{P}(s, T)| \leq L(T) + (6k - 11)$. □

Now, the main result of this chapter is summarized in the following theorem.

Theorem 3.2.1. *In a DGN G_k where k is the network diameter, let the source node be $s = (0, 0)$ and the set of destination nodes be $T = \{t_j = (t_{j_x}, t_{j_y}) | 1 \leq j \leq$*

Table 3.5: Lower and upper bounds of all cases

Case No.	Chosen Cases	Lower Bound	Upper Bound
1	$\langle 1,1,1,1\rangle$	$L(T)$	$L(T)$
2	$\langle 2,0,2,0\rangle$	$L(T)$	$L(T)+(4k-6)$
3	$\langle 2,2,0,0\rangle$	$L(T)+1$	$L(T)+(4k-6)$
4	$\langle 2,1,1,0\rangle$	$L(T)$	$L(T)+(2k-3)$
5	$\langle 2,0,1,1\rangle$	$L(T)+1$	$L(T)+(2k-2)$
6	$\langle 2,1,0,1\rangle$	$L(T)+1$	$L(T)+(2k-3)$
7	$\langle 3,0,0,1\rangle$	$L(T)+2$	$L(T)+(4k-6)$
8	$\langle 3,1,0,0\rangle$	$L(T)+1$	$L(T)+(4k-6)$
9	$\langle 3,0,1,0\rangle$	$L(T)+1$	$L(T)+(4k-5)$
10	$\langle 4,0,0,0\rangle$	$L(T)+2$	$L(T)+(6k-11)$

$4\}$. *Then, there exist NDP* $\mathbb{P}(s,T)$ *such that the sum of the lengths of the NDP in* $\mathbb{P}(s,T)$ *is*

$$L(T) \leq |\mathbb{P}(s,T)| \leq L(T) + (6k-11)$$

Proof. Any DGN G_k can be divided into four non-overlapped quadrants based on the source node's address. These quadrants are Q_N, Q_W, Q_S, and Q_E as defined in Section 3.2.1. The four destination nodes can be distributed in exactly $\binom{4+4-1}{4} = 35$ ways represented as $\langle |Q_N|, |Q_W|, |Q_S|, |Q_E|\rangle$ where $|Q_i|$ is the number of destination nodes in quadrant i for $i = N, W, S, E$. To prove the theorem we need to show that the NDP exist for each one of these 35 cases. However, since G_k is vertex symmetric, constructing the NDP for only 10 cases is equivalent to constructing the NDP for the 35 cases. Table 3.1 shows the chosen 10 cases and the equivalent cases. The total number of these cases is 35. Lemmas 3.2.1 to 3.2.10 prove that the NDP exist for the chosen 10 cases. Table 3.5 shows the upper and lower bounds of these 10 cases. It follows that the sum of lengths of the NDP is $L(T) \leq |\mathbb{P}(s, T)| \leq L(T) + (6k - 11)$. □

3.2.3 Time Complexity

The overall time complexity of the proposed algorithm equals the sum of time complexity of Step 1 and Step 2 (see Algorithm 3). In Step 1, the algorithm counts the number of destination nodes in each quadrant based on the addresses of the source and destination nodes. Clearly, this step can be done in a constant time $O(1)$.

In Step 2, the algorithm constructs the NDP by executing the procedure of one case out of 10 cases based on the number of destination nodes in each quadrant. Thus, the time complexity of Step 2 equals the time complexity of the most time consuming case among the 10 cases.

To construct the NDP, the algorithm needs to know the left, right, top, bottom, max-weight, and min-weight destination nodes of a specific quadrants as defined in Definition 3.2.2. That requires sorting the destination node addresses based on three criteria:

1. the x-coordinate to know the left and right destination nodes,
2. the y-coordinate to know the top and bottom destination nodes, and
3. the weight as defined in Equation 3.2 to know the max-weight, and min-weight destination nodes.

This sorting can be done using the bucket sorting method. In the worst case, the number of elements to be sorted equals four (the number of destination nodes)

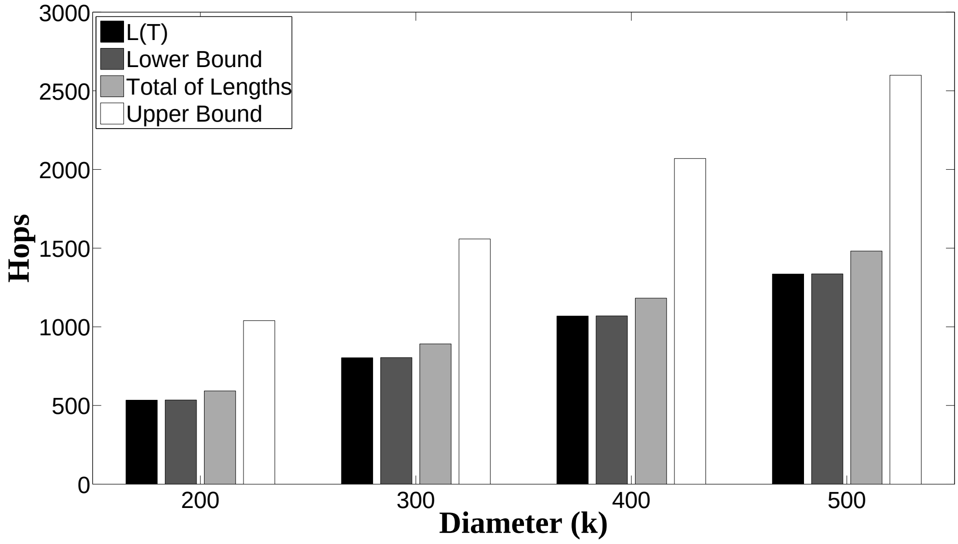


Figure 3.16: Shortest non-NDP vs. actual NDP

and that happens in Case 10. So, the time complexity of Step 2 is equal to $O(3 \cdot 4) = O(1)$. As a result, the overall time complexity of our proposed algorithm is a constant time $O(1)$.

3.3 Algorithm Execution Results

In this section, we show the results of simulating the proposed algorithm. We mainly measure the sum of path lengths $|\mathbb{P}(s, T)|$ and compare it to the sum of destination nodes' weights $L(T)$ and the lower and upper bounds. The sum of destination nodes' weights $L(T)$ is equal to the sum of the shortest paths lengths where these paths are not necessarily node disjoint paths (NDP). Our simulation results show that all of the time the proposed algorithm gives NDP. The results also show that we need on the average about 10% more hops than the sum of destination nodes' weights $L(T)$ to construct the NDP in Gaussian networks.

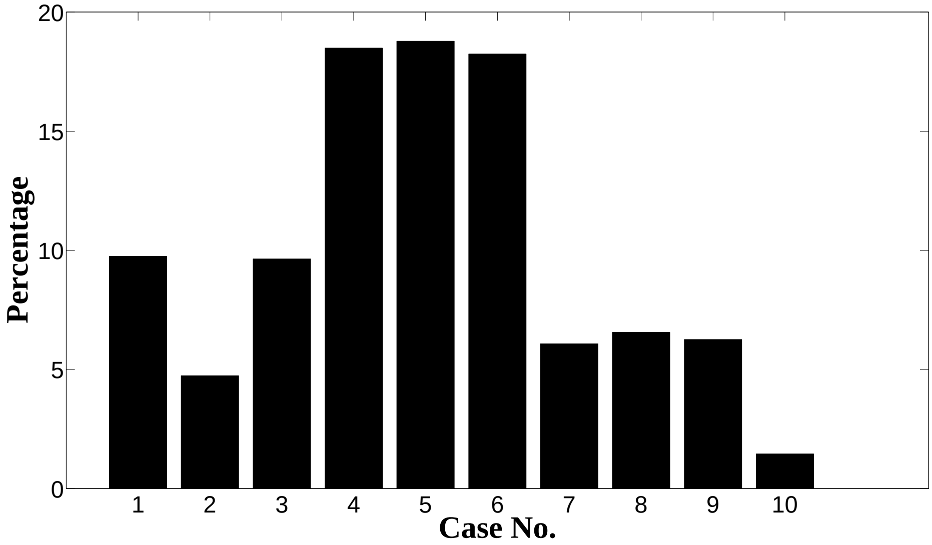


Figure 3.17: Distribution of occurrence ($k = 500$, runs= $10,000$)

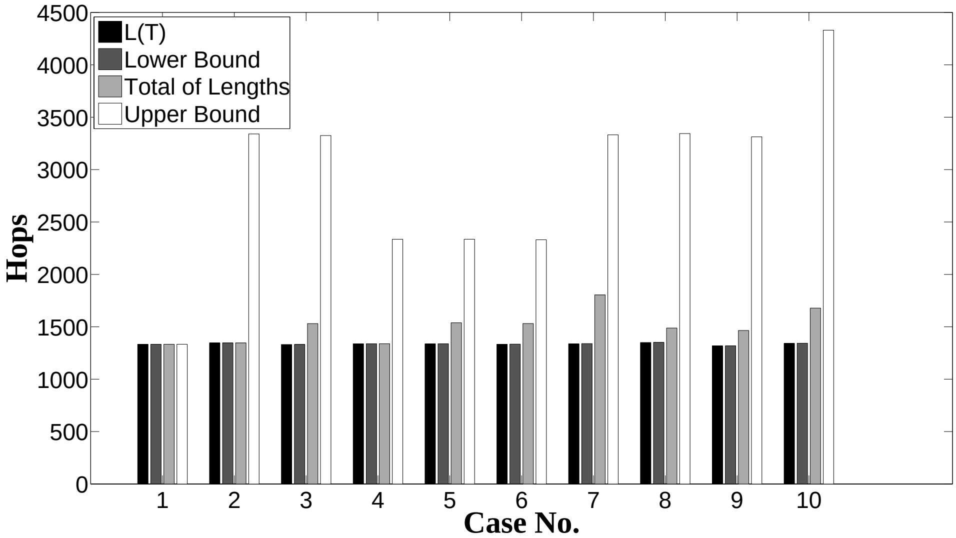


Figure 3.18: Case-wise shortest non-NDP vs. actual NDP ($k = 500$, runs= $10,000$)

We ran a simulator of the proposed algorithm 10,000 times for each one of the following networks: $G_{200}, G_{300}, G_{400}$, and G_{500}. In each run, the simulator *randomly* generated the four destination nodes T and the source node s. It returned the

NDP $\mathbb{P}(s,T)$ for each run. After taking the averages, the results are shown in Figure 3.16. In this figure, we compare the average number of hops of the sum of destination nodes' weights $L(T)$ and the sum of the actual NDP lengths $|\mathbb{P}(s,T)|$ along with the average of the lower and upper bounds. Clearly, the sum of the actual NDP lengths constructed by the proposed algorithm is very close to the sum of destination nodes' weights. In fact, the algorithm can construct the NDP with about 10% more hops on the average than the sum of destination nodes' weights. This result is true regardless of the size of the network because the number of nodes in the network is irrelevant to the NDP construction process in the proposed algorithm.

For more clarification on why the difference between the actual NDP lengths and shortest distances is small, Figure 3.17 shows the distribution of occurrence of each case for G_{500} over 10,000 runs. As shown in this figure, Cases 4, 5, and 6 are the most occurred cases with about 18% each. As shown in Table 3.5, the upper bounds of these cases are less than the other cases' upper bounds (except Case 1). Moreover, Case 10 which has the maximum upper bound occurs the least with 2% occurrence.

For more insights on the results, Figure 3.18 compares for each case between the actual NDP lengths and shortest distances along with the lower and upper bounds for G_{500} over 10,000 runs. First, notice that the sum of the NDP lengths of Cases 1, 2, and 4 is equal to the sum of the shortest paths and this sum is equal to the lower bound. Second, notice that the sum of the NDP lengths is far closer to the lower bound than the upper bound in all cases except Case 1 where the upper bound is same as the lower bound.

3.4 Conclusion

In this chapter we provide and prove an algorithm to construct all NDP from a single source node to a set of destination nodes in the dense Gaussian networks (DGNs). This algorithm constructs four NDP and this is the maximum number of NDP that can be obtained because the degree of the nodes is four. We show that the sum of the NDP lengths constructed by the algorithm is bounded between the sum of the shortest paths and this sum plus $(6k - 11)$ where k is the diameter. We also show that the time complexity of the algorithm is constant $O(1)$. Finally, the algorithm execution results show that on the average the sum of NDP lengths is only about 10% more than the sum of the shortest paths.

Chapter 4: One-to-Many Node Disjoint Paths Routing in Hexagonal Mesh Networks

In this chapter, an efficient constant time complexity algorithm that constructs node disjoint paths (NDP) from a single source node to the maximum number of destination nodes in Hexagonal Mesh Networks (HMNs) is given.

The rest of this chapter is organized as follows: Section 4.1 recalls several preliminaries about HMNs, Section 4.2 describes the proposed routing algorithm, and Section 4.3 concludes this chapter.

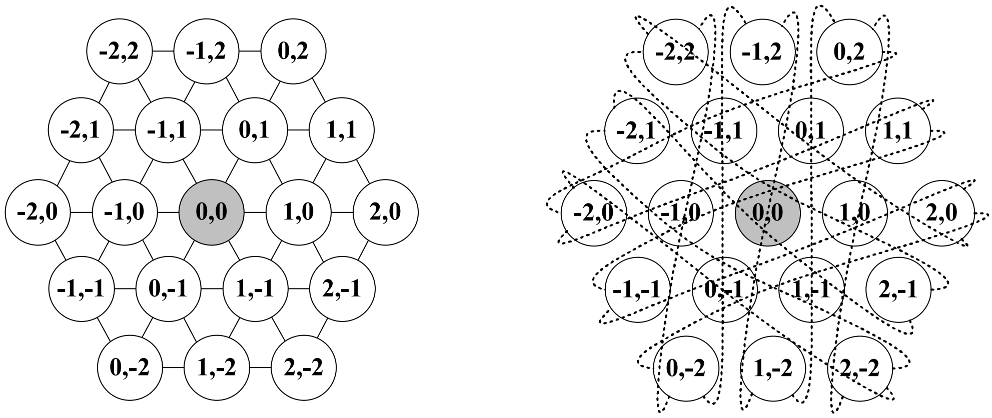
4.1 Hexagonal Mesh Networks Preliminaries

Hexagonal Mesh Networks (HMNs) are defined in terms of Eisenstein-Jacobi (EJ) integers. The following subsections explain the EJ integers, describe HMNs, and formally define the one-to-many node disjoint paths (NDP) routing problem in these networks.

4.1.1 EJ Integers

The set of all EJ integers, $\mathbb{Z}[\rho]$, is defined as $\{x+y\rho|x, y \in \mathbb{Z}\}$ where $\rho = (1+\mathbf{i}\sqrt{3})/2$ and $\mathbf{i} = \sqrt{-1}$.

The set $\mathbb{Z}[\rho]$ is a Euclidean domain and the norm of an EJ integer $\omega = \omega_x + \omega_y\rho$



(a) Regular links

(b) Wraparound links

Figure 4.1: Links in H_2

is defined as [17]:

$$\mathcal{N}(\omega) = {\omega_x}^2 + {\omega_y}^2 + \omega_x\omega_y$$

So, a Euclidean division algorithm for EJ integers exists. Let $\omega_1, \omega_2 \in \mathbb{Z}[\rho]$ and $\omega_2 \neq 0$. Then, there exist $q, r \in \mathbb{Z}[\rho]$ such that $\omega_1 = q\omega_2 + r$ and $\mathcal{N}(r) < \mathcal{N}(\omega_2)$. Let $\alpha = a + b\rho \in \mathbb{Z}[\rho]$ be nonzero where a and b are integers. Then, $\omega_1, \omega_2 \in \mathbb{Z}[\rho]$ are congruent modulo α if there exists $\gamma \in \mathbb{Z}[\rho]$ such that $\omega_2 - \omega_1 = \gamma\alpha$. Congruence and the EJ integers modulo α are denoted by $\omega_2 \equiv \omega_1 (\text{mod } \alpha)$ and $\mathbb{Z}[\rho]_\alpha$ respectively. The number of elements in $\mathbb{Z}[\rho]_\alpha$ is equal to $\mathcal{N}(\alpha) = a^2 + b^2 + ab$ [17].

4.1.2 Hexagonal Mesh Networks

Hexagonal Mesh Networks (HMNs) are two-dimensional networks generated by EJ integers and these were first introduced in [32]. Let $\alpha \in \mathbb{Z}[\rho]$ be nonzero. Each node in a HMN generated by α represents an EJ integer that belongs to the EJ integers modulo α denoted by $\mathbb{Z}[\rho]_\alpha$. So, the number of nodes in this HMN is equal to $\mathcal{N}(\alpha)$. It is proved that for a given diameter $k \in \mathbb{Z}^+$, a HMN achieves the largest network size with $3k^2+3k+1$ nodes when it is generated by $\alpha = (k+1)+k\rho$ [15]. This network is referred as HMN.

In this work, we assume the generator of the HMN is $\alpha = (k+1) + k\rho$ and denote this HMN by H_k where k is the network diameter. Figure 4.1 shows H_2 which is generated by $\alpha = 3 + 2\rho$. In this example, the number of nodes is equal to $\mathcal{N}(3+2\rho) = 3 \times 2^2 + 3 \times 2 + 1 = 19$ and the diameter $k = 2$. In the following, we use this example to explain some properties of HMNs.

Addressing: In this work we use the addressing scheme given in [15]. In this scheme, the address of each node $\omega = \omega_x + \omega_y\rho \in \mathbb{Z}[\rho]_\alpha$ is (ω_x, ω_y), where ω_x and ω_y represents the signed distance from the origin along the horizontal axis (East) and the 60 degrees axis (Northeast), respectively. In Figure 4.1, the 2-tuples inside each node are the addresses.

Connectivity: Two nodes $\omega_1, \omega_2 \in \mathbb{Z}[\rho]_\alpha$ in H_k are connected (neighbors) if and only if $(\omega_1 - \omega_2) \equiv \pm 1, \pm\rho, \pm\rho^2 \pmod{\alpha}$ where $\alpha = (k+1) + k\rho$ is the generator of H_k. So, each node $\omega = \omega_x + \omega_y\rho \in \mathbb{Z}[\rho]_\alpha$ is connected to six neighbors:

1. $\omega^E = (\omega_x + 1) + \omega_y\rho \pmod{\alpha}$,
2. $\omega^{NE} = \omega_x + (\omega_y + 1)\rho \pmod{\alpha}$,
3. $\omega^{NW} = (\omega_x - 1) + (\omega_y + 1)\rho \pmod{\alpha}$,
4. $\omega^W = (\omega_x - 1) + \omega_y\rho \pmod{\alpha}$,
5. $\omega^{SW} = \omega_x + (\omega_y - 1)\rho \pmod{\alpha}$, and
6. $\omega^{SE} = (\omega_x + 1) + (\omega_y - 1)\rho \pmod{\alpha}$

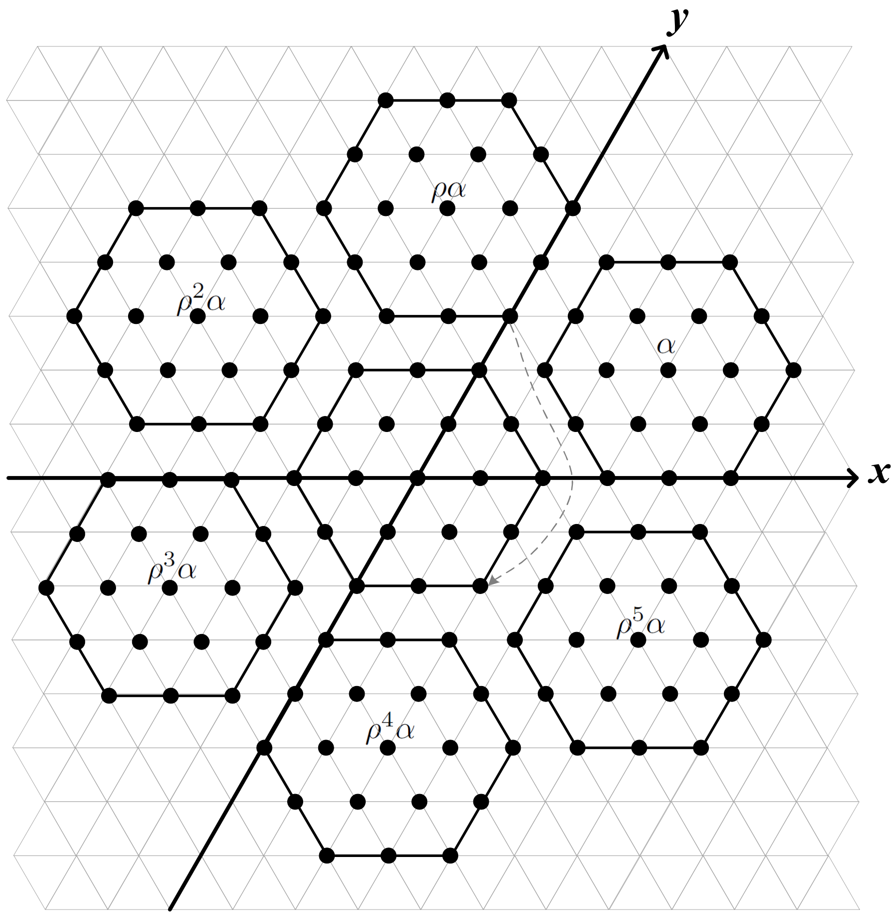


Figure 4.2: Tiling of H_2

where $\omega^N, \omega^W, \omega^S, \omega^E \in \mathbb{Z}[\rho]_\alpha$.

The modulo function (mod α) is used to build the wraparound links. Let $\beta = \beta_x + \beta_y\rho$ be one of the above neighbors before applying the modulo function. Also, let $\beta \notin \mathbb{Z}[\mathbf{i}]_\alpha$ (i.e. β is not one of the network's nodes). So, we need to apply the modulo function to translate β to one of the network's nodes. The modulo function $\beta \pmod{\alpha}$ is given by the following [32]:

$$\begin{aligned} &\beta \pmod{\alpha} = \beta - \hat{\alpha} \\ &\quad \text{where} \quad \hat{\alpha} = \operatorname*{argmin}_{\alpha_i \in A}\{\beta - \alpha_i\} \\ &\quad \text{and} \quad A = \{\alpha, \rho\alpha, \rho^2\alpha, \rho^3\alpha, \rho^4\alpha, \rho^5\alpha\} \end{aligned} \tag{4.1}$$

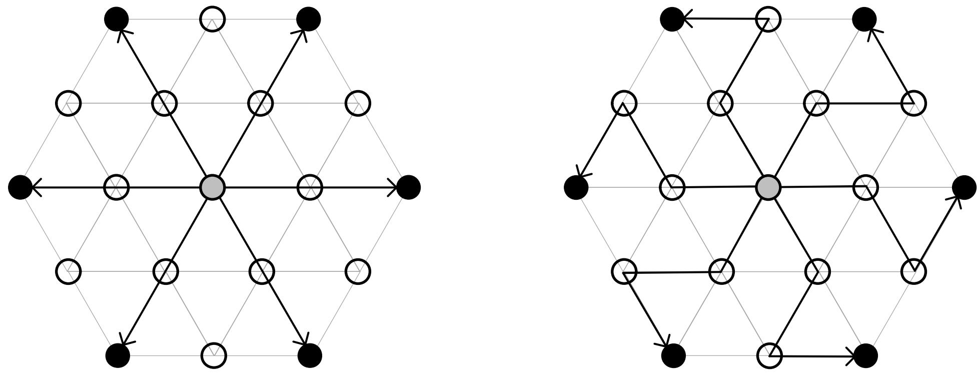
The set A contains the centers of all adjacent hexagons in "the infinite (equilateral) triangle grid in the plan whose nodes are the vertices of regular hexagons of side length one centered at the origin and whose edges are all line segments of length one connecting two nodes" [4]. For example, Figure 4.2 shows the tiling of H_2. In this example, the centers of the adjacent hexagons are $A = \{3+2\rho, -2+5\rho, -5+3\rho, -3-2\rho, 2-5\rho, 5-3\rho\}$.

In Figure 4.1b, the dashed links are the wraparound links built using Equation 4.1 and these wraparound links always connect two border nodes. For example, the NE neighbor of $\omega = 2\rho$ is $\omega^{NW} = 3\rho \pmod{3+2\rho} = (3\rho) - (-2+5\rho) = 2-2\rho$ where $\beta = 3\rho$ (as shown by the dashed arrow in Figure 4.2). Another example is that the W neighbor of $\omega = -2+\rho$ is $\omega^{W} = -3+\rho \pmod{3+2\rho} = (-3+\rho) - (-5+3\rho) = 2-2\rho$ where $\beta = -3+\rho$.

Diameter: The diameter is the largest possible distance between any two nodes in a network. The diameter of H_k is equal to k [32]. For example in Figure 4.1, the diameter of H_2 is equal to two.

Degree: The node degree is the number of its neighbors. In HMNs, each node is adjacent to six other nodes. So, the node degree is equal to six for all nodes [15, 32].

Path: A path from node ω_1 to node ω_2 is denoted by $P(\omega_1, \omega_2) = \langle \omega_1, a_1, a_2, \ldots, a_{|P(\omega_1,\omega_2)|-1}, \omega_2 \rangle$ where $|P(\omega_1, \omega_2)|$ is the length of the path and each two consecutive



(a) (b)

Figure 4.3: Different examples of NDP in H_3

nodes (e.g. ω_1 and a_1) are neighbors. Sometimes, we write the path $P(\omega_1, \omega_2)$ as $\omega_1 \to a_1 \to a_2 \to \cdots \to \omega_2$. If ω_1 and ω_2 are not neighbors, we write $\omega_1 \overset{dir}{\Rightarrow} \omega_2$ to denote a straight path from ω_1 to ω_2 in direction $dir \in \{E, NE, NW, W, SW, SE\}$. For example in Figure 4.1, $(2, -1) \overset{NW}{\Rightarrow} (-1, 2)$ denotes the path $(2, -1) \to (1, 0) \to (0, 1) \to (-1, 2)$.

One-to-Many NDP: Given a source node s and a set of distinct destination nodes $T = \{t_1, t_2, \ldots, t_6\}$, where $s \notin T$, a set of one-to-many NDP connects s to each destination node t_j, $j \in \{1, 2, \ldots, 6\}$, and satisfy the condition that the only common node among all paths is the source node s. Since the degree of each node in HMN equals six, the maximum number of destination nodes for which a set of NDP can be obtained from a given source node also equals six and this is the case in this work.

For a particular s and T, there are more than one possible set of NDP from s to T. One of these possible sets is denoted by $\mathbb{P}(s, T)$. For example consider the network in Figure 4.1, let the source node be $s = (0, 0)$ and the set of destination nodes be $T = \{(0, 2), (-2, 2), (-2, 0), (0, -2), (2, -2), (2, 0)\}$. Then, two different possible sets of NDP are given in Figure 4.3.

The following section describes our routing algorithm from the source node s to each of the six destination nodes in T using NDP.

4.2 One-to-Many Node Disjoint Paths Routing

A hexagonal mesh network H_k can be partitioned into six non-overlapped sectors based on the source node's address. Definition 4.2.1 defines these sectors.

Definition 4.2.1. *In a hexagonal mesh network H_k where k is the diameter, let the source node be $s = (s_x, s_y)$. Then, Sector 1, Sector 2, Sector 3, Sector 4, Sector 5, and Sector 6 are respectively defined as follows:*

1. $S_1 = \{(x, y) \in H_k \mid (x \geq s_x) \wedge (y > s_y)\}$
2. $S_2 = \{(x, y) \in H_k \mid (x < s_x) \wedge (y > s_y) \wedge |x| \leq y\}$
3. $S_3 = \{(x, y) \in H_k \mid (x < s_x) \wedge (y \geq s_y) \wedge |x| > y\}$
4. $S_4 = \{(x, y) \in H_k \mid (x \leq s_x) \wedge (y < s_y)\}$
5. $S_5 = \{(x, y) \in H_k \mid (x > s_x) \wedge (y < s_y) \wedge x \leq |y|\}$
6. $S_6 = \{(x, y) \in H_k \mid (x > s_x) \wedge (y \leq s_y) \wedge x > |y|\}$

We can easily show that the number of nodes in each sector equals $k(k+1)/2$.

In case the source node s is $(0, 0)$ (as we assume in this work), the sectors are defined as follows (see Figure 4.4):

1. $S_1 = \{(x, y) \in H_k \mid (x \geq 0) \wedge (y > 0)\}$

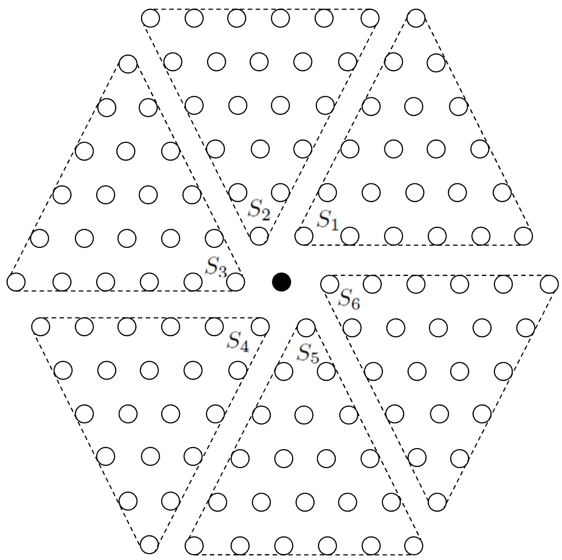


Figure 4.4: Sectors in H_6

2. $S_2 = \{(x, y) \in H_k \mid (x < 0) \wedge (y > 0) \wedge |x| \leq y\}$

3. $S_3 = \{(x, y) \in H_k \mid (x < 0) \wedge (y \geq 0) \wedge |x| > y\}$

4. $S_4 = \{(x, y) \in H_k \mid (x \leq 0) \wedge (y < 0)\}$

5. $S_5 = \{(x, y) \in H_k \mid (x > 0) \wedge (y < 0) \wedge x \leq |y|\}$

6. $S_6 = \{(x, y) \in H_k \mid (x > 0) \wedge (y \leq 0) \wedge x > |y|\}$

For the hexagonal mesh network H_6 as shown in Figure 4.4, the number of nodes in each sector equals $6(6 + 1)/2 = 21$ nodes where $k = 6$.

Figure 4.5 shows the tiling and sectors of H_2. Figure 4.6 shows the sectors connected to each border node in S_1. From these two figures, it is important to notice the following for better understanding of the proposed algorithm:

1. Each sector is connected to all other sectors through one or more nodes. For example in Figure 4.6, S_1 is connected to all other sectors.

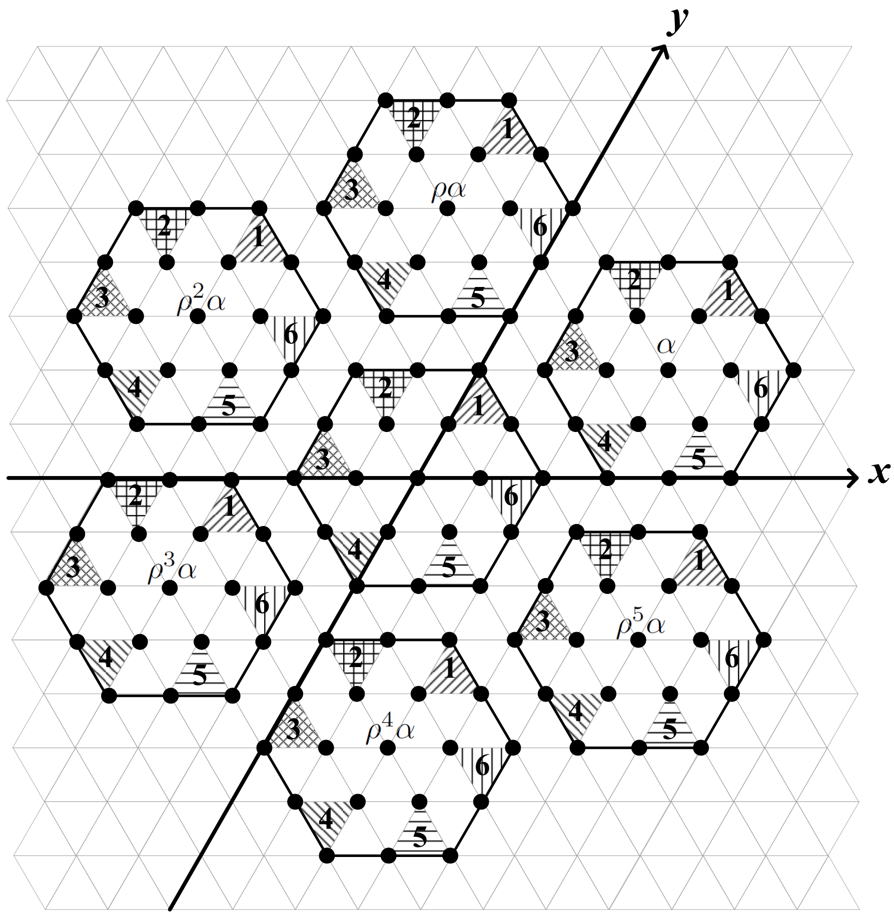


Figure 4.5: Tiling of H_2

2. Sector S_i, $i = 1, 2, \ldots, 6$, is connected to: 1) sector $S_{i+2 \pmod 6}$ through exactly two nodes, and 2) sector $S_{i+5 \pmod 6}$ through exactly one node. For example in Figure 4.5, S_1 is connected to: 1) S_3 through $(0, 2)$ and $(1, 1)$, and 2) S_5 through $(0, 2)$.

3. The node that connects S_i to $S_{i+5 \pmod 6}$ is also used to connect S_i to $S_{i+2 \pmod 6}$. For example in Figure 4.5, $(0, 2)$ is used to connect S_1 to both S_3 and S_5.

Each sector is a triangle with three sides. In this work, S_i^r denotes a side of sec-

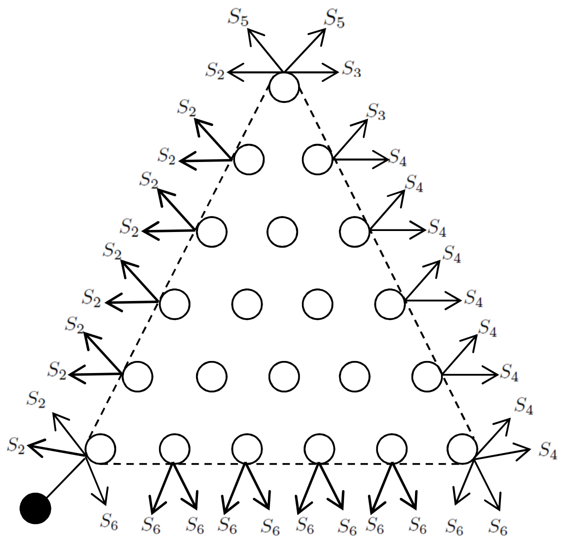


Figure 4.6: Sectors connected to S_1 (H_6)

tor S_i specified by direction $r \in \{E, N, W, S\}$ where $i \in \{1, 2, \ldots, 6\}$. For example in Figure 4.4, S_1^S denotes the south side of Sector one (i.e. $(0,1), (1,1), (2,1), \ldots, (k-1,1)$). The sides of Sector one are S_1^E, S_1^W, and S_1^S.

If node $\omega \notin S_i^r$, then $\omega \overset{dir}{\Rightarrow} S_i^r$ denotes a straight path from ω to the sector side S_i^r in the direction dir. Note that there is one and only one node that is part of this path and this side. For example in Figure 4.4, $(2,2) \overset{NW}{\Rightarrow} S_1^W$ denotes the path starting from $(2,2)$ and ending in S_1^W going in the NW direction. This path is $(2,2) \to (1,3) \to (0,4)$ where $(0,4) \in S_1^W$. Similarly, if node $\omega \notin S_i^r$ but one of its neighbors is in S_i^r, then we write $\omega \to S_i^r$.

After partitioning the network into six sectors, we propose an algorithm that constructs six node disjoint paths (NDP) $\mathbb{P}(s, T)$ from the source node s to the six destination nodes in $T = \{t_j = (t_{j_x}, t_{j_y}) | 1 \leq j \leq 6\}$ where $s \notin T$. The algorithm consists of two main parts: rotation and construction.

The rotation part rotates the network in the clock-counter direction six times.

In each time, sector S_i becomes $S_{i+1 \pmod 6}$. Each rotation is performed by multiplying all nodes in the network by ρ.

The construction part constructs the NDP from the source node to whatever destination nodes in sector S_1 according to the following cases:

- Case 1: S_1 contains six destination nodes.
- Case 2: S_1 contains five destination nodes.
- Case 3: S_1 contains four destination nodes.
- Case 4: S_1 contains three destination nodes.
- Case 5: S_1 contains two destination nodes.
- Case 6: S_1 contains one destination node.

In this section, we explain how to construct the NDP for each one of these cases. In Section 4.2.1, we explain how the algorithm uses them. Before that, we need the following definitions.

Definition 4.2.2. *In a hexagonal mesh network H_k, where k is the diameter, let the source node be $(0,0)$. Then, the E, NE, NW, W, SW, and SE NDP start with $(0,0) \to (1,0)$, $(0,0) \to (0,1)$, $(0,0) \to (-1,1)$, $(0,0) \to (-1,0)$, $(0,0) \to (0,-1)$, and $(0,0) \to (1,-1)$ respectively.*

Definition 4.2.3. *In a hexagonal mesh network H_k where k is the diameter, let $t_j = (t_{j_x}, t_{j_y}) \in S_i$ for $j, i = 1, 2, \ldots, 6$ be any destination node. Then, the destination node t_j is:*

- *the top destination node of S_i if $t_{j_y} = \max\{t_{r_y} | t_r = (t_{r_x}, t_{r_y}) \in S_i\}$,*

- *the bottom destination node of* S_i *if* $t_{j_y} = \min\{t_{r_y} | t_r = (t_{r_x}, t_{r_y}) \in S_i\}$,
- *the left destination node of* S_i *if* $t_{j_x} = \min\{t_{r_x} | t_r = (t_{r_x}, t_{r_y}) \in S_i\}$,
- *the right destination node of* S_i *if* $t_{j_x} = \max\{t_{r_x} | t_r = (t_{r_x}, t_{r_y}) \in S_i\}$,
- *the max-weight destination node of* S_i *if* $W(t_j) = \max\{W(t_r) | t_r = (t_{r_x}, t_{r_y}) \in S_i\}$, *and/or*
- *the min-weight destination node of* S_i *if* $W(t_j) = \min\{W(t_r) | t_r = (t_{r_x}, t_{r_y}) \in S_i\}$.

Note that the top, bottom, left, right, max-weight, or min-weight destination nodes as defined in Definition 4.2.3 are not necessarily unique. So, we say, for example, top/2$^{\text{nd}}$ left of S_i to uniquely specify a destination node by choosing the most 2$^{\text{nd}}$ left destination node among those top destination nodes in case the top destination node is not unique.

Now, we explain how to construct the NDP for each case.

Case 1: Six destination nodes in S_1

In this case, six destination nodes t_j, where $j = 1, 2, \ldots, 6$, exist in sector S_1. Theorem 4.2.1 explains the process of constructing the node disjoint paths (NDP) from the source node s to these destination nodes.

Theorem 4.2.1. *In a hexagonal mesh network* H_k *where* k *is the network diameter, let the source node be* $s = (0, 0)$ *and the set of destination nodes be* $T = \{t_j = (t_{j_x}, t_{j_y}) | 1 \leq j \leq 6\}$ *such that* $t_j \in S_1$. *Then, there exist NDP* $\mathbb{P}(s, T)$.

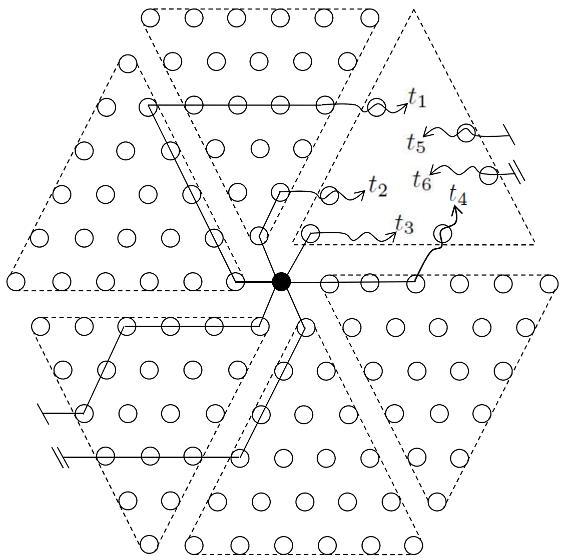


Figure 4.7: NDP outside Sector 1 in Case 1

Proof. To construct the NDP from the source node s to the six destination nodes in S_1, the algorithm connects two destination nodes (say, t_1 and t_2) to S_2, one destination node (say, t_3) to the source's neighbor $(0,1)$, one destination node (say, t_4) to S_6, and two destination nodes (say, t_5 and t_6) to S_4. These paths have two portions: inside and outside S_1. The following paths are the portions outside S_1 (see Figure 4.7):

- Assuming the border node in S_2^E that is connected to t_1 is on top of the border node in S_2^E that is connected to t_2, the path to t_1 is $t_1 \rightarrow \cdots \rightarrow S_1^W \overset{W}{\Rightarrow} S_3^E \overset{SE}{\Rightarrow} (-1,0) \rightarrow s$. If node $(0,k)$ is used in $t_1 \rightarrow \cdots \rightarrow S_1^W$, the path is $t_1 \rightarrow \cdots \rightarrow S_1^W \rightarrow S_2^E \overset{W}{\Rightarrow} S_2^W \rightarrow S_3^E \overset{SE}{\Rightarrow} (-1,0) \rightarrow s$.

- The path to t_2 is $t_2 \rightarrow \cdots \rightarrow S_1^W \rightarrow S_2^E \overset{SW}{\Rightarrow} (-1,1) \rightarrow s$.

- The path to t_4 is $t_4 \rightarrow \cdots \rightarrow S_1^S \rightarrow S_6^N \overset{W}{\Rightarrow} (1,0) \rightarrow s$.

- Assuming the border node in S_4^W that is connected to t_5 is on top of the

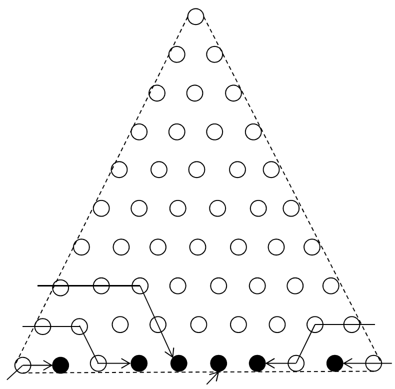


Figure 4.8: Example of Case 1.1 (H_{10})

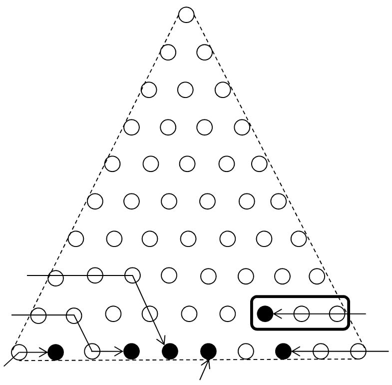
border node in S_4^W that is connected to t_6, the path to t_5 is $t_5 \rightarrow \cdots \rightarrow S_1^E \rightarrow S_4^W \overset{NE}{\Rightarrow} S_4^N \overset{E}{\Rightarrow} (0, -1) \rightarrow s$.

- The path to t_6 is $t_6 \rightarrow \cdots \rightarrow S_1^E \rightarrow S_4^W \overset{E}{\Rightarrow} S_5^W \overset{NE}{\Rightarrow} (1, -1) \rightarrow s$.

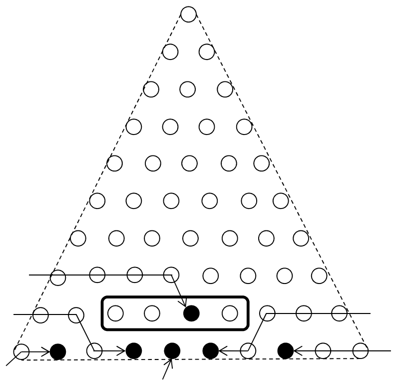
The previous NDP are the paths outside S_1. Next we show the NDP inside S_1. Basically, we need to connect two destination nodes to S_2^E, one destination node to S_6^N, two destination nodes to S_4^W, and one destination node to the source's neighbor $(0, 1)$. The process of constructing the paths inside S_1 depends on the destination node locations as follows:

Case 1.1 (Six destination nodes have $y = 1$):

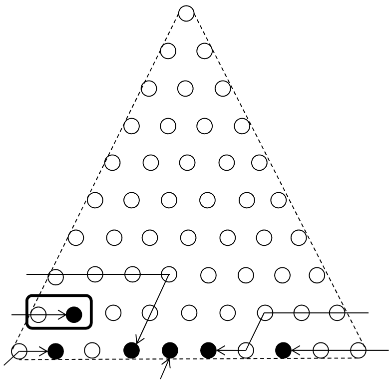
In this case, all destination nodes are in S_1^S. The NDP are as follows: the 1$^{\text{st}}$ left destination node is connected to node $(0, 1)$; the 2$^{\text{nd}}$ and 3$^{\text{rd}}$ left destination nodes are connected to S_2; the 4$^{\text{th}}$ left destination node is connected to S_6; and the 5$^{\text{th}}$ and 6$^{\text{th}}$ left destination nodes are connected to S_4. These paths are straightforward and can be immediately gleaned from the example shown in Figure 4.8.



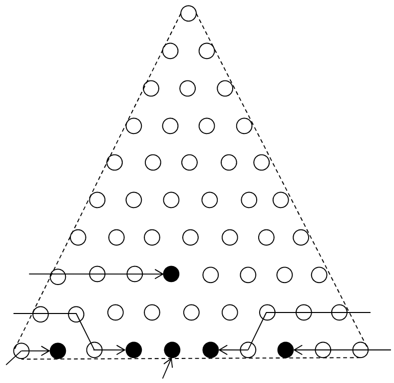
(a) Case 1.2.1



(b) Case 1.2.2



(c) Case 1.2.3



(d) Case 1.2.4

Figure 4.9: Examples of Case 1.2 (H_{10})

Case 1.2 (Five destination nodes have $\boldsymbol{y} = 1$):

Let the destination node that its y coordinate equals or greater than two be $\hat{t} = (\hat{t}_x, \hat{t}_y)$. Only among the destination nodes other than $\hat{t}$, let the 1st left destination node be $t_L = (t_{L_x}, 1)$ and the 1st right destination node be $t_R = (t_{R_x}, 1)$. Then, the NDP for this case depend on the location of $\hat{t}$ as follows :

Case 1.2.1 ($\hat{t}_y = 2$ and $\hat{t}_x \geq t_{R_x} - 1$): In this case, $\hat{t}$ is connected to S_4. The NDP to the remaining destination nodes are as follows: the 1st left destination node is connected to node $(0, 1)$; the 2nd and 3rd left destination nodes are connected to S_2; the 4th left destination node is connected to S_6; and the 5th left destination node (among the ones) is connected to S_4. These paths are straightforward and can be immediately gleaned from the example shown in Figure 4.9a where the marked area represents all possibilities of $\hat{t}$.

Case 1.2.2 ($\hat{t}_y = 2$ and $t_{L_x} < \hat{t}_x < t_{R_x} - 1$): In this case, $\hat{t}$ is connected to S_2. The NDP to the remaining destination nodes are as follows: among the ones in S_1^S, the 1st left destination node is connected to node $(0, 1)$; the 2nd left destination node is connected to S_2; the 3rd left destination node is connected to S_6; and the 4th and 5th left destination nodes are connected to S_4. These paths are straightforward and can be immediately gleaned from the example shown in Figure 4.9b.

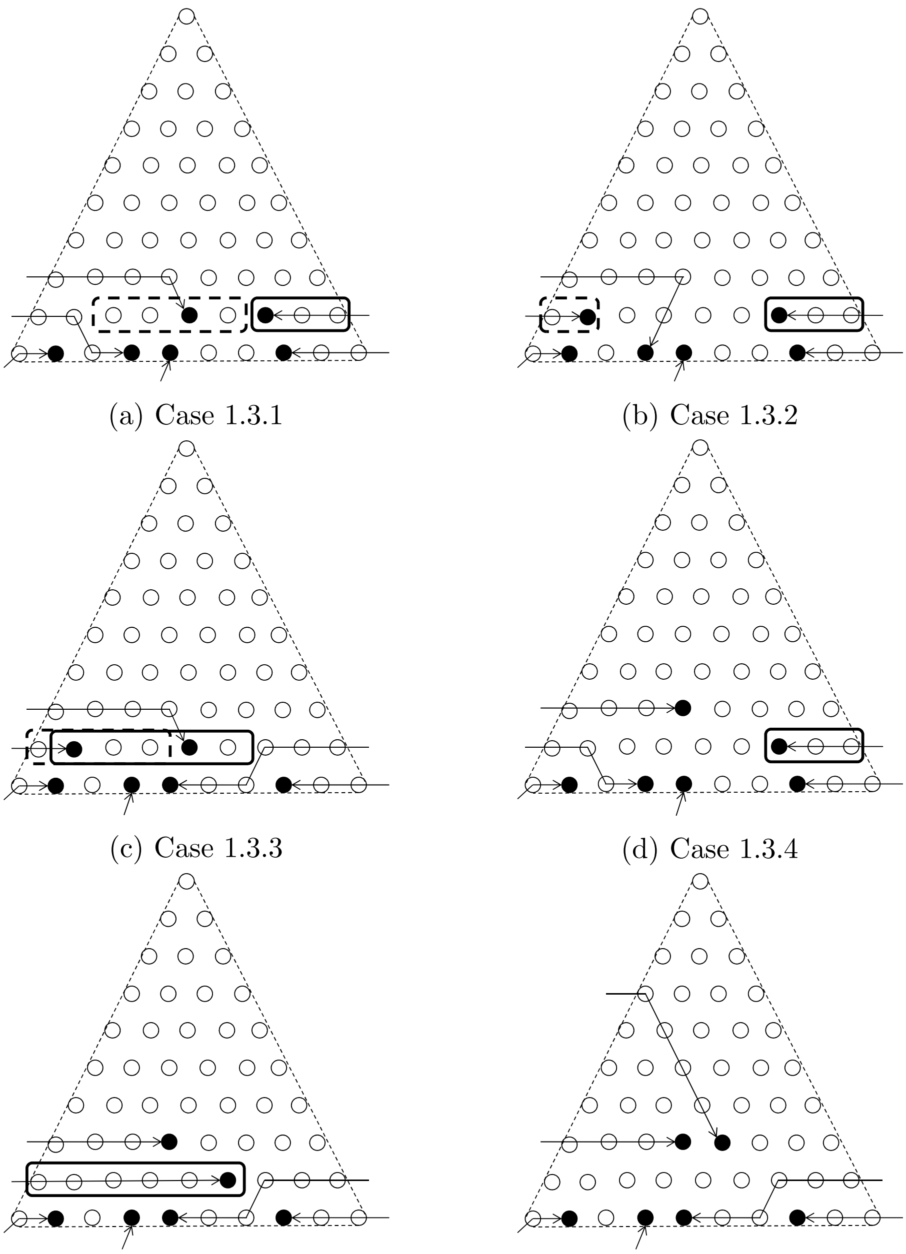
Case 1.2.3 ($\hat{t}_y = 2$ and $\hat{t}_x \leq t_{L_x}$): In this case, each destination node is connected to the same sector as Case 1.2.2. However, the NDP are slightly different. These paths are straightforward and can be immediately gleaned from the example shown in Figure 4.9c.

Case 1.2.4 ($\hat{t}_y > 2$): Also in this case, each destination node is connected to the same sector as Case 1.2.2. However, the NDP are slightly different. These paths are straightforward and can be immediately gleaned from the example shown in Figure 4.9d.

That covers all possibilities of $\hat{t}$.

Case 1.3 (Four destination nodes have $y = 1$):

Let the two destination nodes that their y coordinates equals or greater than



(a) Case 1.3.1 (b) Case 1.3.2

(c) Case 1.3.3 (d) Case 1.3.4

(e) Case 1.3.5 (f) Case 1.3.6

Figure 4.10: Examples of Case 1.3 (H_{10})

two be $\hat{t}_1 = (\hat{t}_{1_x}, \hat{t}_{1_y})$ and $\hat{t}_2 = (\hat{t}_{2_x}, \hat{t}_{2_y})$. Among the destination nodes other than $\hat{t}_1$ and $\hat{t}_2$, let the 1^{st} left destination node be $t_L = (t_{L_x}, 1)$ and the 1^{st}

right destination node be $t_R = (t_{R_x}, 1)$. Then, the NDP for this case depend on the locations of $\hat{t}_1$ and $\hat{t}_2$ as follows :

Case 1.3.1 ($\hat{t}_{1_y} = \hat{t}_{2_y} = 2$ and $\hat{t}_{1_x} > \hat{t}_{2_x}$ and $\hat{t}_{1_x} \geq t_{R_x} - 1$ and $\hat{t}_{2_x} > t_{L_x}$): In this case, two destination nodes are on the second bottom row (see Figure 4.10a). Among them, the x coordinate of the 1st right destination node is equal or grater then the x coordinate minus one of the 1st right destination node among the destination nodes in row $y = 1$. It follows that we cannot connect the 2nd right destination node in the first bottom row to sector S_4 through the second bottom row. However, we can connect the 2nd left destination node in the first bottom row to sector S_2 through the second bottom row because of $\hat{t}_{2_x} > t_{L_x}$. Same reasoning is applied for the following cases of Case 1.3.

In Case 1.3.1, the NDP are as follows: $\hat{t}_1$ and $\hat{t}_2$ are connected to S_4 and S_2 respectively. Among the remaining destination nodes, the 1st left destination node is connected to node $(0, 1)$; the 2nd left destination nodes is connected to S_2; the 3rd destination node is connected to S_6; and the 4th left destination node is connected to S_4. These paths are straightforward and can be immediately gleaned from the example shown in Figure 4.10a.

Case 1.3.2 ($\hat{t}_{1_y} = \hat{t}_{2_y} = 2$ and $\hat{t}_{1_x} > \hat{t}_{2_x}$ and $\hat{t}_{1_x} \geq t_{R_x} - 1$ and $\hat{t}_{2_x} \leq t_{L_x}$): In this case, each destination node is connected to the same sector as Case 1.3.1. However, the NDP are slightly different. These paths are straightforward and can be immediately gleaned from the example shown in Figure 4.10b.

Case 1.3.3 ($\hat{t}_{1_y} = \hat{t}_{2_y} = 2$ and $\hat{t}_{1_x} > \hat{t}_{2_x}$ and $\hat{t}_{1_x} < t_{R_x} - 1$): In this case, $\hat{t}_1$ and $\hat{t}_2$ are connected to S_2. Among the remaining destination nodes, the 1$^{\text{st}}$ left destination node is connected to node $(0, 1)$; the 2$^{\text{nd}}$ left destination node is connected to S_6; the 3$^{\text{rd}}$ and the 4$^{\text{th}}$ left destination nodes are connected to S_4. These paths are straightforward and can be immediately gleaned from the example shown in Figure 4.10c.

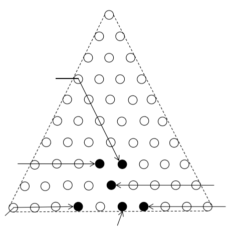
Case 1.3.4 ($\hat{t}_{1_y} = 2$ and $\hat{t}_{2_y} > 2$ and $\hat{t}_{1_x} \geq t_{R_x} - 1$): In this case, $\hat{t}_1$ is connected to S_4. $\hat{t}_2$ is connected to S_2. Among the remaining destination nodes, the 1$^{\text{st}}$ left destination node is connected to node $(0, 1)$; the 2$^{\text{nd}}$ left destination node is connected to S_2; the 3$^{\text{rd}}$ left destination node is connected to S_6; and the 4$^{\text{th}}$ left destination node is connected to S_4. These paths are straightforward and can be immediately gleaned from the example shown in Figure 4.10d.

Case 1.3.5 ($\hat{t}_{1_y} = 2$ and $\hat{t}_{2_y} > 2$ and $\hat{t}_{1_x} < t_{R_x} - 1$): In this case, each destination node is connected to the same sector as Case 1.3.3. However, the NDP are slightly different. These paths are straightforward and can be immediately gleaned from the example shown in Figure 4.10e.

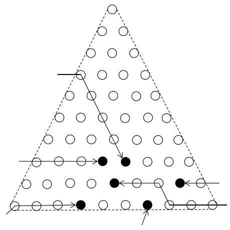
Case 1.3.6 ($\hat{t}_{1_y}, \hat{t}_{2_y} > 2$): In this case, each destination node is connected to the same sector as Case 1.3.3. However, the NDP are slightly different. These paths are straightforward and can be immediately gleaned from the example shown in Figure 4.10f.

Case 1.4 (Three or less destination nodes have $y = 1$):

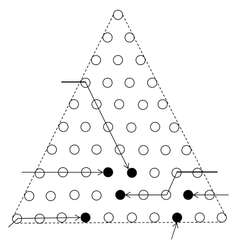
In this case, the following steps construct the NDP (Figure 4.11a shows an example.):



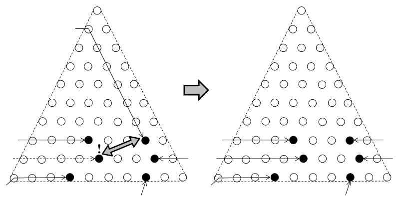
(a) Case 1.4 (E)



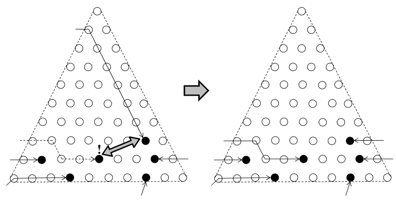
(b) Case 1.4 (SE)



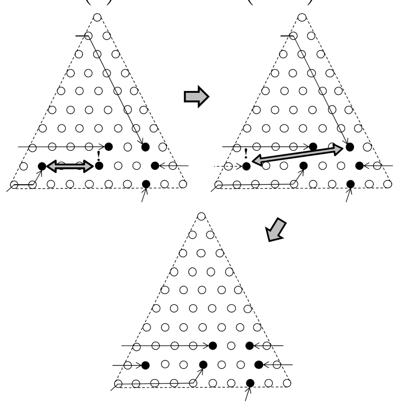
(c) Case 1.4 (NE)



(d) Case 1.4 (W)



(e) Case 1.4 (NW)



(f) Case 1.4 (SW)

Figure 4.11: Examples of Case 1.4 (H_{10})

1. Among all destination nodes, find the *top/2nd left* destination node . Let it be t_1. The path to t_1 is $t_1 \overset{NW}{\Rightarrow} S_1^W$. Now, t_1 is connected to S_2.

2. Excluding t_1, find the *top/left* destination node. Let it be t_2. The path

to t_2 is $t_2 \overset{W}{\Rightarrow} S_1^W$. Now, t_2 is connected to S_2.

3. Excluding t_1 and t_2, find the *min-weight/top* destination node. Let it be t_3. The path to t_3 is $t_3 \overset{SW}{\Rightarrow} S_1^S \overset{W}{\Rightarrow} (0,1) \rightarrow s$.

4. Excluding t_1, t_2, and t_3, find the *min-weight/bottom* destination node. Let it be t_4. The path to t_4 is $t_4 \overset{SE}{\Rightarrow} S_1^S$. Now, t_4 is connected to S_6.

5. Excluding t_1, t_2, t_3, and t_4, find the *max-weight/bottom* destination node. Let it be t_5. The path to t_5 is $t_5 \overset{E}{\Rightarrow} S_1^E$. Now, t_5 is connected to S_4.

   Clearly, the previous NDP are visible since we make sure that there is no destination node on the way of the constructed path. For example, t_5 is the *max-weight/bottom* destination node. It follows that there is no destination node from t_5 to the border node on the same row on the east direction.

6. Let the remaining destination node be t_6. To construct the path to t_6, we check the availability of the following paths in the following order (if a path is not available because one or more of its nodes have been used by the previous paths constructed in the above steps, we go to the next path):

   (a) Check the availability of the following path that connects t_6 to S_4: $t_6 \overset{E}{\Rightarrow} S_1^E$. (Figure 4.11a shows an example.) If this path is not available, then the destination node that blocks this path must be t_5 because its weight is more than the weight of t_6.

   (b) Check the availability of the following path that connects t_6 to S_4: $t_6 \overset{E}{\Rightarrow} t_5^W \rightarrow t_5^{SW} \overset{E}{\Rightarrow} S_1^E$. (Figure 4.11b shows an example.)

(c) Check the availability of the following path that connects t_6 to S_4: $t_6 \overset{E}{\Rightarrow} t_5^W \rightarrow t_5^{NE} \overset{E}{\Rightarrow} S_1^E$. (Figure 4.11c shows an example.)

(d) Check the availability of the following path that connects t_6 to S_2: $t_6 \overset{W}{\Rightarrow} S_1^W$. (Figure 4.11d shows an example, the dashed path.) If this path is available, then there are three destination nodes connected to S_2 (i.e. t_1, t_2, and t_6) while there is only one destination node connected to S_4 (i.e. t_5). To fix this, we switch between t_6 and t_1 by connecting t_1 to S_4 using the following path: $t_1 \overset{E}{\Rightarrow} S_1^E$. This path must be available. Otherwise, one of the previous paths must be available.

If the above path to t_6 is not available, then the destination node that blocks this path must be t_2 because it is the *top/left* destination node among all nodes except t_1.

(e) Check the availability of the following path that connects t_6 to S_2: $t_6 \overset{W}{\Rightarrow} t_2^E \rightarrow t_2^{NW} \overset{W}{\Rightarrow} S_1^W$. (Figure 4.11e shows an example.) For the same reason, we have to switch between t_6 and t_1 same as before.

(f) If none of the previous paths is available, then the following steps construct the path (Figure 4.11f shows an example.):

    i. Connect t_6 to $(0, 1)$ (instead of t_3) using the following path: $t_6 \overset{SW}{\Rightarrow} S_1^S \overset{W}{\Rightarrow} (0, 1) \rightarrow s$. This path must be available because the only destination node that can blocks it is t_6 which is blocking one of the previous paths.

    ii. Connect t_3 (the one that was connected to $(0, 1)$) to S_2 using the following path: $t_3 \overset{W}{\Rightarrow} S_1^W$. Now, there are three destination nodes connected to S_2 (i.e. t_1, t_2, and t_3) while there is only one destination node connected to S_4 (i.e. t_5).

iii. Connect t_1 to S_4 (instead of S_2) same as before.

This covers all possible cases and completes the proof. □

Case 2: Five destination nodes in S_1

In this case, five destination nodes exist in sector S_1. Theorem 4.2.2 shows the NDP for this case.

Theorem 4.2.2. *In a hexagonal mesh network H_k where k is the network diameter, let the source node be $s = (0,0)$ and the set of destination nodes be $T = \{t_j = (t_{j_x}, t_{j_y}) | 1 \leq j \leq 6\}$ such that five destination nodes exist in S_1. Then, there exist NDP $\mathbb{P}(s,T)$.*

Proof. In this case, one destination node does not exist in S_1. Let it be $t_{\bar{S}_1}$. Then, we have the following cases based on which sector contains $t_{\bar{S}_1}$:

Case 2.1 ($t_{\bar{S}_1}$ in S_6):

The solution of this case is similar to the solution of Case 1 except we remove the step that connects one of the destination node to S_6. The details are as follows:

- If there are five, four, and three destination nodes that their $y = 1$, then the NDP are given in Figure 4.8, Figure 4.9, and Figure 4.10b except that we remove the destination node that is connected to S_6.

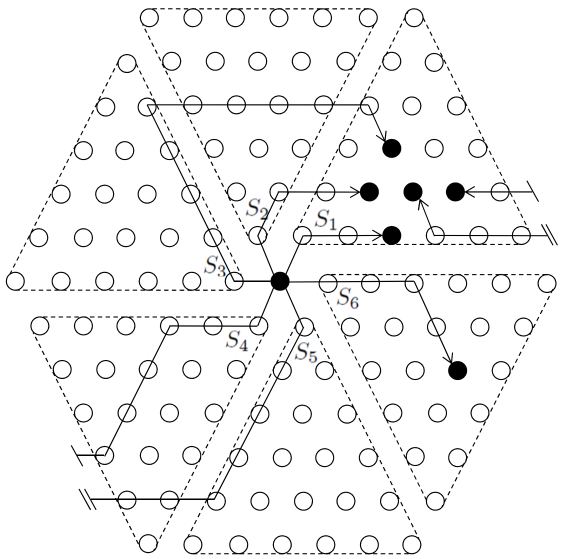


Figure 4.12: Example of Case 2.1 (H_6)

- If there are two or less destination nodes that their $y = 1$, then the NDP are obtained by applying the algorithm given for Case 1.4 except that we remove Step 4 which is the step that connects one of the destination nodes to S_6.

The NDP outside S_1 are exactly as given in Case 1 except that the path from the source node to $t_{\bar{S}_1}$ within S_6 is slightly different. This path is straightforward and can be immediately gleaned from the example of Case 2.1 shown in Figure 4.12. In this example, one destination node in S_1 has $y = 1$. So, we apply the algorithm given in Case 1.4 except that we remove Step 6.

Case 2.2 ($t_{\bar{S}_1}$ in S_4 or S_5):

Before we show the NDP for this case, it is important to notice from Figure 4.6 that each border node in S_1^E has two neighbors in S_4 except node

$(1, k)$ which has only one neighbor in S_4 (i.e. $(-(k-1), -1)$). So, we can always connect one destination node from S_1 to S_4 as long as: 1) $t_{\bar{S}_1}$ is not $(-(k-1), -1)$, or 2) the border node in S_1 that is connected to this destination node is not $(1, k-1)$. Since $t_{\bar{S}_1}$ exists in either S_4 or S_5, the NDP outside S_1 are exactly as given in Case 1 except that we connect only one destination node to S_4. If the above conditions are satisfied, then this is a special case and we will show its solution later. Otherwise, we connect the border node to whatever available of its neighbor. Then we connect the source node to the *top/right* (among the destination node in S_4 (if any) and the border node along the path to one of the destination node in S_1) by going from the source to $(0, -1)$, then going west to the same x-coordinate as the *top/right* node,then going south-west to this node. The other node in S_4 is connected to S_5 as explained in Case 1.

Based on the number of destination nodes that have $y = 1$, we have the following cases:

Case 2.2.1 (Five destination nodes have $y = 1$):

The NDP for this case are straightforward and can be immediately gleaned from Figure 4.13.

Case 2.2.2 (Four destination nodes have $y = 1$):

In this case, the y-coordinate of one of the destination nodes in S_1 does not equal to one. Let this destination node be $\hat{t}$. Among the destination nodes that have $y = 1$, let the left destination node be t_L. If $\hat{t}_y = 2$ and $\hat{t}_x \leq t_{L_x}$, then the NDP are given in Figure 4.14a. Otherwise, the NDP are given in Figure 4.14b.

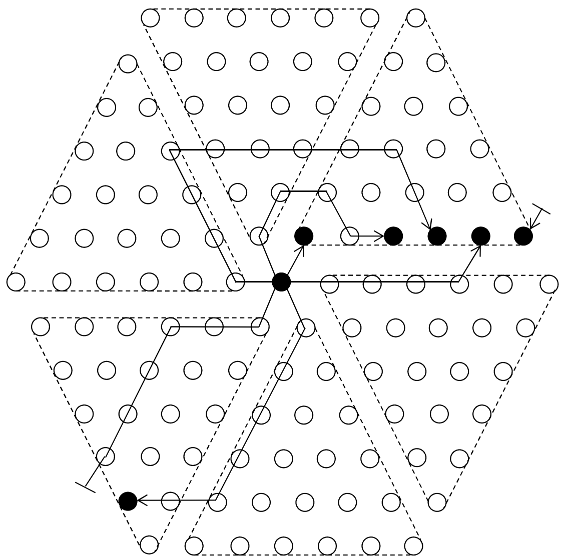


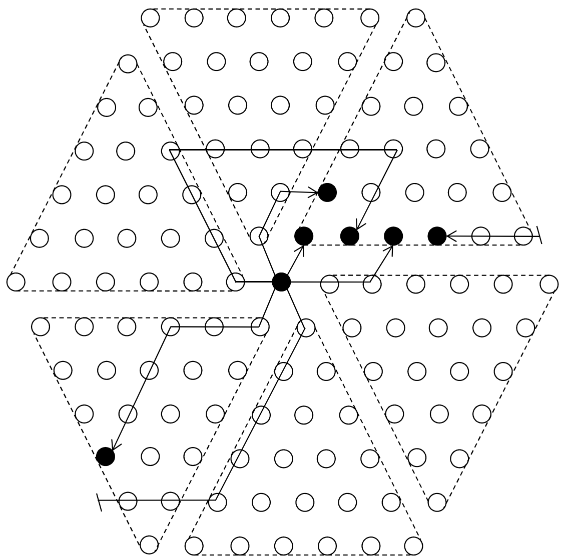
Figure 4.13: Example of Case 2.2.1 (H_6)

Case 2.2.3 (Three or less destination nodes have $y = 1$):

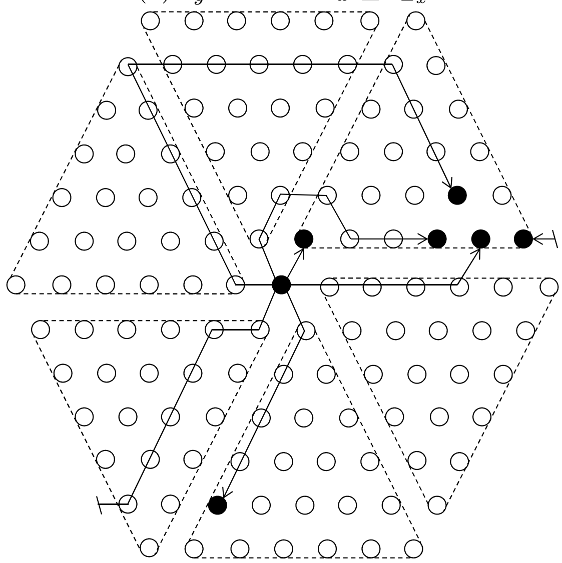
The NDP for this case are constructed by the following algorithm:

1. Apply Steps 1 to 5 from the algorithm given for Case 1.4 on the five destination nodes in S_1. As a result of Step 5, one of these destination nodes is connected to a border node that is adjacent to S_4. Let this border node be c_5^1.

2. If $c_5^1 = (1, k-1)$ *and* $t_{\bar{S}_1} = (-(k-1), -1)$, then we cannot connect c_5^1 to S_4 because the only neighbor of c_5^1 in S_4 is $t_{\bar{S}_1}$. Note that in this case, $(0, k)$ and $(0, k-1)$ must be destination nodes (otherwise $c_5^1 \neq (1, k-1)$). To construct the NDP in this case we connect $(0, k)$ to S_5, $(0, k-1)$ to S_2, $c_5^1 = (1, k-1)$ to S_3, the min-weight/top (out of the remaining two destination nodes) to $(0, 1)$, and the last destination node to S_6. Figure 4.15a shows an example of this case where all paths can be immediately gleaned.



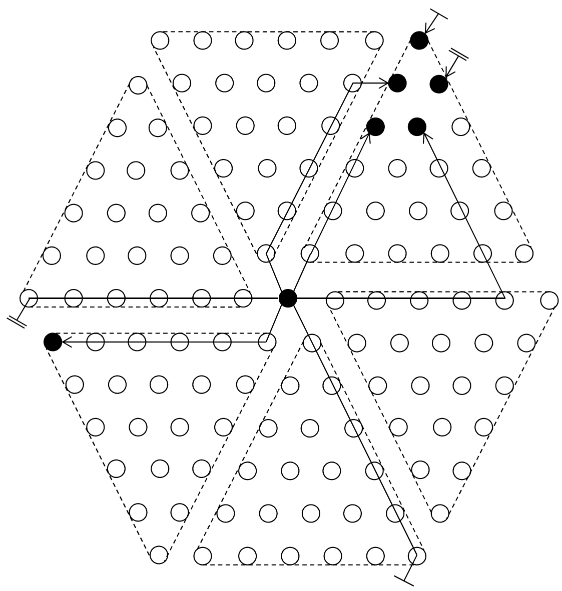
(a) $\hat{t}_y = 2$ and $\hat{t}_x \leq t_{L_x}$



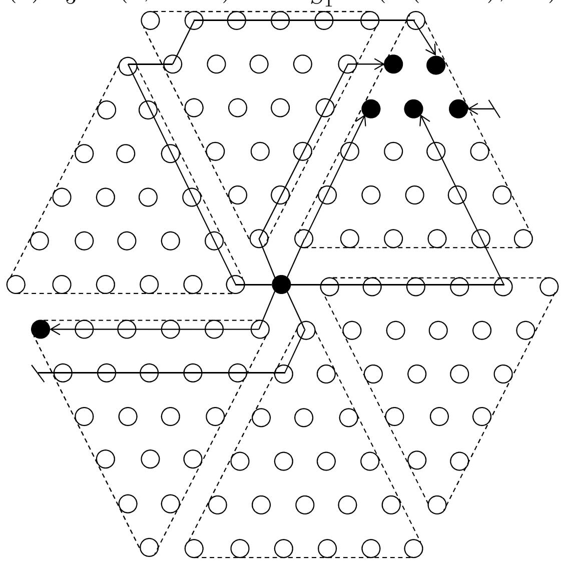
(b) Otherwise

Figure 4.14: Examples of Case 2.2.2 (H_6)

3. If $c_5^1 \neq (1, k-1)$ *or* $t_{\bar{S}_1} \neq (-(k-1), -1)$, then we can safely construct the NDP as given in the algorithm for Case 1.4 by applying the first five steps. Figure 4.15b shows an example of this case where all paths can be immediately gleaned.



(a) $c_5^1 = (1, k-1)$ and $t_{\bar{S}_1} = (-(k-1), -1)$



(b) Otherwise

Figure 4.15: Example of Case 2.2.3 (H_6)

Case 2.3 ($t_{\bar{S}_1}$ in S_2 or S_3): Note that node $(0, k)$ in S_1 is adjacent to exactly one node in S_2 (i.e. $(-1, k)$). Moreover, node $(0, k)$ is not adjacent to S_4. So, if $(0, k)$ is a destination node (or a border node along the path to

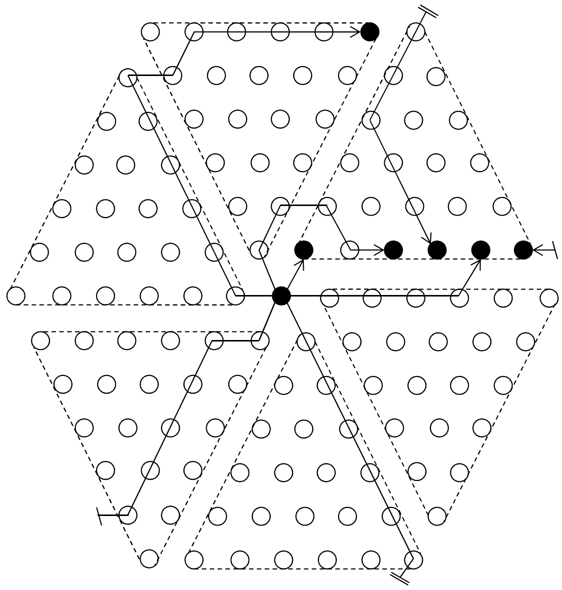


Figure 4.16: Example of Case 2.3.1 (H_6)

a destination node) and $(-1, k)$ is also a destination node, then we cannot connect $(0, k)$ to either S_2 nor S_4. To avoid this case, we connect one of the destination nodes in S_1 to S_5 through the node $(0, k)$. It follows that we need to connect only one destination node to S_4. Since $t_{\bar{S}_1}$ is in S_2 or S_3, we need to connect only one destination node to S_2.

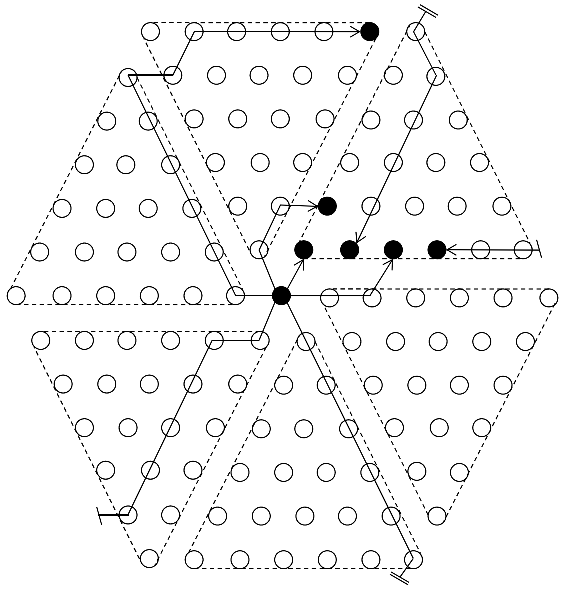
Based on the number of destination nodes that have $y = 1$, we have the following cases:

Case 2.3.1 (Five destination nodes have $y = 1$):

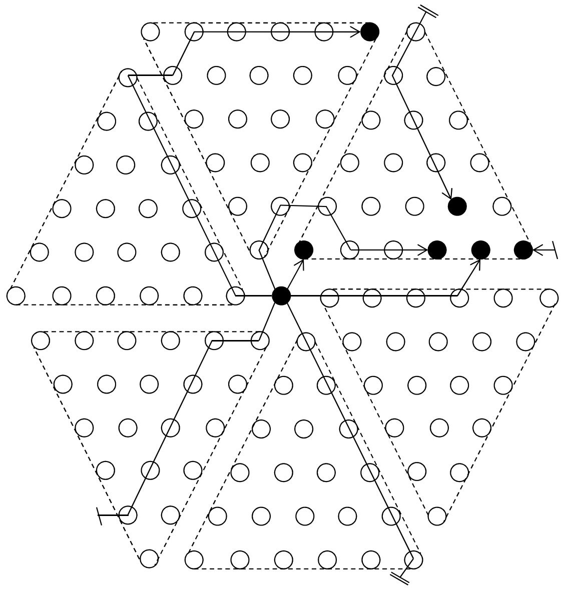
The NDP for this case are very similar to Case 2.2.1 except that we connect one of the destination node to S_5. Figure 4.16 shows an example where all paths can be immediately gleaned.

Case 2.3.2 (Four destination nodes have $y = 1$):

Similar to Case 2.2.1, Case 2.3.2 is divided into two cases based on the



(a) $\hat{t}_y = 2$ and $\hat{t}_x \leq t_{L_x}$



(b) Otherwise

Figure 4.17: Example of Case 2.3.2 (H_6)

location of the destination node that is not on $y = 1$. Figure 4.17 shows both cases where all paths can be immediately gleaned.

Case 2.3.3 (Three or less destination nodes have $y = 1$):

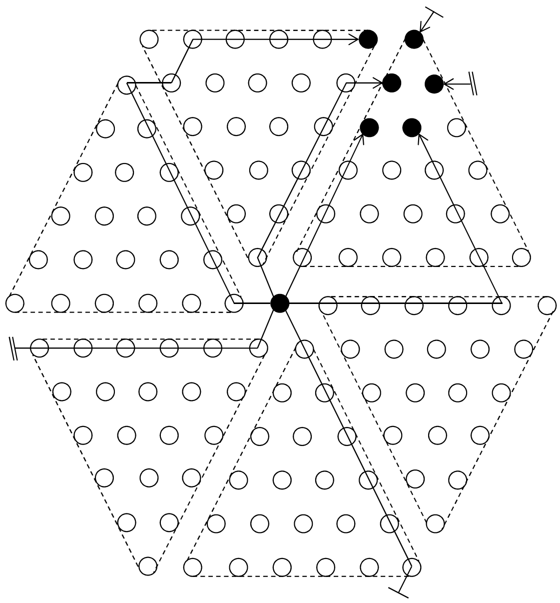


Figure 4.18: Example of Case 2.3.3 (H_6)

To construct the NDP for this case, we apply the algorithm given in Case 1.4 with the following changes:

1. In Step 1 instead of connecting the *top/left* destination node (i.e t_1) to S_2, we connect this destination node to S_5 through node $(0, k)$ using the following path: $t_1 \overset{NW}{\Rightarrow} S_1^W \overset{NE}{\Rightarrow} (0, k) \rightarrow (k, -k) \overset{NW}{\Rightarrow} (1, -1) \rightarrow s$.
2. Remove Step 6 that connects one of the destination nodes to S_4.

Figure 4.18 shows an example of this case.

This covers all possible cases and completes this proof. □

Case 3: Four destination nodes in S_1

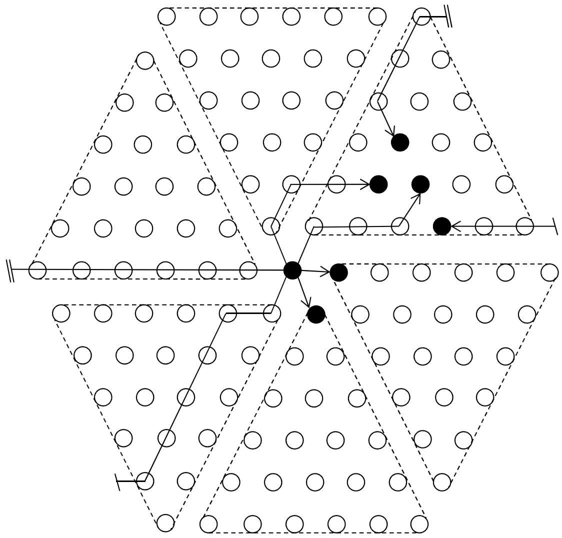


Figure 4.19: Example of Case 3.1 (H_6)

In this case, four destination nodes exist in sector S_1. Theorem 4.2.3 shows the NDP for this case.

Theorem 4.2.3. *In a hexagonal mesh network H_k where k is the network diameter, let the source node be $s = (0, 0)$ and the set of destination nodes be $T = \{t_j = (t_{j_x}, t_{j_y}) | 1 \leq j \leq 6\}$ such that four destination nodes exist in S_1. Then, there exist NDP $\mathbb{P}(s, T)$.*

Proof. In this case, two destination nodes do not exist in S_1. It follows that at least three sectors (out of $S_2, S_3, \ldots, S_6$) do not have any destination nodes. Let (S_a, S_b, S_c) denote these three sectors where $a, b, c \in \{2, \ldots, 6\}$. Then, there are 10 cases in the form of (S_a, S_b, S_c). Assuming that at most two destination nodes have $y = 1$, Table 4.1 lists all cases and provides the NDP for each case. If there are three or four destination nodes that have $y = 1$, then the NDP can be easily constructed.

The following steps show how to construct the NDP using Table 4.1:

Case No.	Condition (S_a, S_b, S_c)	Destination Node		Node Disjoint Path
		out of	t_i	
3.1	(S_2, S_3, S_4)	all	top/2nd left	$t_i \xRightarrow{NW} S_1^W \xRightarrow{NE} (0,k) \xRightarrow{E} (-1,0) \rightarrow s$
		rest	top/left	$t_i \xRightarrow{W} S_2^E \xRightarrow{SW} (-1,1) \rightarrow s$
		rest	min-weight/top	$t_i \xRightarrow{SW} S_1^S \xRightarrow{W} (0,1) \rightarrow s$
		rest	the last one	$t_i \xRightarrow{E} S_4^W \xRightarrow{NE} S_4^N \xRightarrow{E} (0,-1) \rightarrow s$
3.2	(S_2, S_3, S_5)	Look at Table 4.2		
3.3	(S_2, S_3, S_6)	all	top/2nd left	$t_i \xRightarrow{NW} S_1^W \xRightarrow{NE} (0,k) \xRightarrow{E} (-1,0) \rightarrow s$
		rest	top/left	$t_i \xRightarrow{W} S_2^E \xRightarrow{SW} (-1,1) \rightarrow s$
		rest	min-weight/top	$t_i \xRightarrow{SW} S_1^S \xRightarrow{W} (0,1) \rightarrow s$
		rest	the last one	$t_i \xRightarrow{SE} S_6^N \xRightarrow{W} (1,0) \rightarrow s$
3.4	(S_2, S_4, S_5)	all	top/2nd left	$t_i \xRightarrow{NW} S_1^W \xRightarrow{NE} S_5^E \xRightarrow{NW} (1,-1) \rightarrow s$
		rest	top/left	$t_i \xRightarrow{W} S_2^E \xRightarrow{SW} (-1,1) \rightarrow s$
		rest	min-weight/top	$t_i \xRightarrow{SW} S_1^S \xRightarrow{W} (0,1) \rightarrow s$
		rest	the last one	$t_i \xRightarrow{E} S_4^W \xRightarrow{NE} S_4^N \xRightarrow{E} (0,-1) \rightarrow s$
3.5	(S_2, S_4, S_6)	all	top/left	$t_i \xRightarrow{W} S_2^E \xRightarrow{SW} (-1,1) \rightarrow s$
		rest	min-weight/top	$t_i \xRightarrow{SW} S_1^S \xRightarrow{W} (0,1) \rightarrow s$
		rest	min-weight/bottom	$t_i \xRightarrow{SE} S_6^N \xRightarrow{W} (1,0) \rightarrow s$
		rest	the last one	$t_i \xRightarrow{E} S_4^W \xRightarrow{NE} S_4^N \xRightarrow{E} (0,-1) \rightarrow s$
3.6	(S_2, S_5, S_6)	all	top/2nd left	$t_i \xRightarrow{NW} S_1^W \xRightarrow{NE} S_5^E \xRightarrow{NW} (1,-1) \rightarrow s$
		rest	top/left	$t_i \xRightarrow{W} S_2^E \xRightarrow{SW} (-1,1) \rightarrow s$
		rest	min-weight/top	$t_i \xRightarrow{SW} S_1^S \xRightarrow{W} (0,1) \rightarrow s$
		rest	the last one	$t_i \xRightarrow{SE} S_6^N \xRightarrow{W} (1,0) \rightarrow s$
3.7	(S_3, S_4, S_5)	Look at Table 4.4		
3.8	(S_3, S_4, S_6)	all	top/left	$t_i \xRightarrow{NW} S_1^W \xRightarrow{NE} (0,k) \xRightarrow{E} (-1,0) \rightarrow s$
		rest	min-weight/top	$t_i \xRightarrow{SW} S_1^S \xRightarrow{W} (0,1) \rightarrow s$
		rest	min-weight/bottom	$t_i \xRightarrow{SE} S_6^N \xRightarrow{W} (1,0) \rightarrow s$
		rest	the last one	$t_i \xRightarrow{E} S_4^W \xRightarrow{NE} S_4^N \xRightarrow{E} (0,-1) \rightarrow s$
3.9	(S_3, S_5, S_6)	Look at Table 4.5		
3.10	(S_4, S_5, S_6)	all	top/2nd left	$t_i \xRightarrow{NW} S_1^W \xRightarrow{NE} S_5^E \xRightarrow{NW} (1,-1) \rightarrow s$
		rest	min-weight/top	$t_i \xRightarrow{SW} S_1^S \xRightarrow{W} (0,1) \rightarrow s$
		rest	min-weight/bottom	$t_i \xRightarrow{SE} S_6^N \xRightarrow{W} (1,0) \rightarrow s$
		rest	the last one	$t_i \xRightarrow{E} S_4^W \xRightarrow{NE} S_4^N \xRightarrow{E} (0,-1) \rightarrow s$

Table 4.1: Subcases of Case 3: four destination nodes in S_1

Case No.	Conditions (AND) 1	2	3	Destination Node out of	t_i	Node Disjoint Path
3.2.1	S_1^E does not contain dest. nodes			all	top/2$^{\text{nd}}$ left	$t_i \overset{NW}{\Rightarrow} S_1^W \overset{W}{\Rightarrow} S_3^E \overset{SE}{\Rightarrow} (-1,0) \to s$
				rest	top/left	$t_i \overset{W}{\Rightarrow} S_2^E \overset{SW}{\Rightarrow} (-1,1) \to s$
				rest	min-weight/bottom	$t_i \overset{SW}{\Rightarrow} S_1^S \overset{W}{\Rightarrow} (0,1) \to s$
				rest	the last one	$t_i \overset{E}{\Rightarrow} S_1^E \overset{NW}{\Rightarrow} (0,k) \to (k,-k) \overset{NW}{\Rightarrow} (1,-1) \to s$
3.2.2.1	S_1^E contains one dest. node	at most one dest. node has $y=1$			the one in S_1^E	$t_i \overset{NW}{\Rightarrow} (0,k) \to (k,-k) \overset{NW}{\Rightarrow} (1,-1) \to s$
				rest	top/2$^{\text{nd}}$ left	$t_i \overset{NW}{\Rightarrow} S_1^W \overset{W}{\Rightarrow} S_3^E \overset{SE}{\Rightarrow} (-1,0) \to s$
				rest	top/left	$t_i \overset{W}{\Rightarrow} S_2^E \overset{SW}{\Rightarrow} (-1,1) \to s$
				rest	the last one	$t_i \overset{SW}{\Rightarrow} S_1^S \overset{W}{\Rightarrow} (0,1) \to s$
3.2.2.2.1		exactly two dest. nodes have $y=1$	S_4 contains two dest. nodes		the one on S_1^E	$t_i \overset{NW}{\Rightarrow} (0,k) \to (k,-k) \overset{NW}{\Rightarrow} (1,-1) \to s$
				rest	top	$t_i \overset{W}{\Rightarrow} S_2^E \overset{SW}{\Rightarrow} (-1,1) \to s$
				rest	left	$t_i \overset{W}{\Rightarrow} (0,1) \to s$
				rest	the last one	$t_i \overset{SW}{\Rightarrow} S_6^N \overset{W}{\Rightarrow} (1,-1) \to s$
					the ones in S_4	To connect one dest. node to S_3 from S_4, rotate; apply Case 5.5 (S_6) from Table 4.8; and rotate back.
3.2.2.2.2			S_4 contains one dest. node		the one in S_1^E	$t_i \overset{NW}{\Rightarrow} (0,k) \to (k,-k) \overset{NW}{\Rightarrow} (1,-1) \to s$
				rest	top	$t_i \overset{W}{\Rightarrow} S_2^E \overset{SW}{\Rightarrow} (-1,1) \to s$
				rest	left	$t_i \overset{W}{\Rightarrow} (0,1) \to s$
				rest	the last one	If $(-1,-(k-1))$ is available, the path is $t_i \overset{E}{\Rightarrow} (k-1,1) \to (-1,-(k-1))$. Otherwise, the path is $t_i \overset{E}{\Rightarrow} (k-1,1) \to (-2,-(k-2))$. Then, apply Case 5.5 ($S_6$) (after rotation) on this dest. node and the one in S_4 to connect one of them to S_3.
3.2.2.2.3			S_4 does not contain dest. nodes		the one in S_1^E	$t_i \overset{NW}{\Rightarrow} (0,k) \to (k,-k) \overset{NW}{\Rightarrow} (1,-1) \to s$
				rest	top	$t_i \overset{W}{\Rightarrow} S_2^E \overset{SW}{\Rightarrow} (-1,1) \to s$
				rest	left	$t_i \overset{W}{\Rightarrow} (0,1) \to s$
				rest	the last one	$t_i \overset{E}{\Rightarrow} (k-1,1) \to (-1,-(k-1))$ / To connect one dest. node to S_3, rotate; apply Case 5.5 (S_6) on $(-1,-(k-1))$ and $(0,-k)$; and rotate back.
					the ones in S_6	To connect one dest. node to S_4, rotate; apply Case 5.4 (S_5); and rotate back.
3.2.3.1	S_1^E contains two dest. nodes	S_4 contains two dest. nodes		S_1^E	top	$t_i \overset{NW}{\Rightarrow} (0,k) \to (k,-k) \overset{NW}{\Rightarrow} (1,-1) \to s$
					bottom	$t_i \overset{SE}{\Rightarrow} S_6^N \overset{W}{\Rightarrow} (1,0) \to s$
				rest	top/2$^{\text{nd}}$left	$t_i \overset{NW}{\Rightarrow} S_1^W \to S_2^E \overset{SW}{\Rightarrow} (-1,1) \to s$
				rest	the last one	$t_i \overset{SW}{\Rightarrow} S_1^S \overset{W}{\Rightarrow} (0,1) \to s$
					the ones in S_4	To connect one dest. node to S_3, rotate; apply Case 5.5 (S_6); and rotate back.
3.2.3.2		S_4 contains one dest. node		S_1^E	top	$t_i \overset{NW}{\Rightarrow} (0,k) \to (k,-k) \overset{NW}{\Rightarrow} (1,-1) \to s$
					bottom	If $(-1,-(k-1))$ is available, the path is $t_i \overset{SE}{\Rightarrow} (k-1,1) \to (-1,-(k-1))$. Otherwise, the path is $t_i \overset{SE}{\Rightarrow} (k-1,1) \to (-2,-(k-2))$. Then, apply Case 5.5 ($S_6$) (after rotation) on this dest. node and the one in S_4 to connect one of them to S_3.
				rest	top/2$^{\text{nd}}$left	$t_i \overset{NW}{\Rightarrow} S_1^W \to S_2^E \overset{SW}{\Rightarrow} (-1,1) \to s$
				rest	the last one	$t_i \overset{SW}{\Rightarrow} S_1^S \overset{W}{\Rightarrow} (0,1) \to s$
3.2.3.3		S_4 does not contain dest. nodes		S_1^E	top	$t_i \overset{NW}{\Rightarrow} (0,k) \to (k,-k) \overset{NW}{\Rightarrow} (1,-1) \to s$
					bottom	$t_i \overset{SE}{\Rightarrow} (k-1,1) \to (-1,-(k-1))$ / To connect one dest. node to S_3, rotate; apply Case 5.5 (S_6) on $(-1,-(k-1))$ and $(0,-k)$; and rotate back.
					the ones in S_6	To connect one dest. node to S_4, rotate; apply Case 5.4 (S_5); and rotate back.
				rest	top/2$^{\text{nd}}$left	$t_i \overset{NW}{\Rightarrow} S_1^W \to S_2^E \overset{SW}{\Rightarrow} (-1,1) \to s$
				rest	the last one	$t_i \overset{SW}{\Rightarrow} S_1^S \overset{W}{\Rightarrow} (0,1) \to s$

Continue on Table 4.3

Table 4.2: Subcases of Case 3.2: (S_2, S_3, S_5)

Case No.	Conditions (AND) 1	2	3	Destination Node out of	t_i	Node Disjoint Path	
3.2.4.1	S_1^E contains three dest. nodes	S_4 contains two dest. nodes		S_1^E	top	$t_i \overset{NW}{\Rightarrow} (0,k) \to (k,-k) \overset{NW}{\Rightarrow} (1,-1) \to s$	
					bottom	$t_i \overset{SE}{\Rightarrow} S_6^N \overset{W}{\Rightarrow} (1,0) \to s$	
				rest	bottom/right	$t_i \overset{SW}{\Rightarrow} S_1^S \overset{W}{\Rightarrow} (0,1) \to s$	
				rest	the last one	$t_i \overset{W}{\Rightarrow} S_2^E \overset{SW}{\Rightarrow} (-1,1) \to s$	
					the ones in S_4	To connect one dest. node to S_3, rotate; apply Case 5.5 (S_6); and rotate back.	
3.2.4.2		S_4 contains one dest. node		S_1^E	top	$t_i \overset{NW}{\Rightarrow} (0,k) \to (k,-k) \overset{NW}{\Rightarrow} (1,-1) \to s$	
					bottom	If $(-1,-(k-1))$ is available, the path is $t_i \overset{SE}{\Rightarrow} (k-1,1) \to (-1,-(k-1))$. Otherwise, the path is $t_i \overset{SE}{\Rightarrow} (k-1,1) \to (-2,-(k-2))$. Then, apply Case 5.5 ($S_6$) (after rotation) on this dest. node and the one in S_4 to connect one of them to S_3.	
				rest	bottom/right	$t_i \overset{SW}{\Rightarrow} S_1^S \overset{W}{\Rightarrow} (0,1) \to s$	
				rest	the last one	$t_i \overset{W}{\Rightarrow} S_2^E \overset{SW}{\Rightarrow} (-1,1) \to s$	
3.2.4.3		S_4 does not contain dest. nodes		S_1^E	top	$t_i \overset{NW}{\Rightarrow} (0,k) \to (k,-k) \overset{NW}{\Rightarrow} (1,-1) \to s$	
					bottom	$t_i \overset{SE}{\Rightarrow} (k-1,1) \to (-1,-(k-1))$	To connect one dest. node to S_3, rotate; apply Case 5.5 (S_6) on $(-1,-(k-1))$ and $(0,-k)$; and rotate back.
					the ones in S_6	To connect one dest. node to S_4, rotate; apply Case 5.4 (S_5); and rotate back.	
				rest	bottom/right	$t_i \overset{SW}{\Rightarrow} S_1^S \overset{W}{\Rightarrow} (0,1) \to s$	
				rest	the last one	$t_i \overset{W}{\Rightarrow} S_2^E \overset{SW}{\Rightarrow} (-1,1) \to s$	
3.2.5	S_1^E contains four dest. nodes			all	top	$t_i \overset{NW}{\Rightarrow} (0,k) \to (k,-k) \overset{NW}{\Rightarrow} (1,-1) \to s$	
				rest	2nd top	$t_i \overset{W}{\Rightarrow} S_3^E \overset{SE}{\Rightarrow} (-1,0) \to s$	
				rest	3rd top	$t_i \overset{W}{\Rightarrow} S_2^E \overset{SW}{\Rightarrow} (-1,1) \to s$	
				rest	the last one	$t_i \overset{SW}{\Rightarrow} S_1^S \overset{W}{\Rightarrow} (0,1) \to s$	

Table 4.3: Subcases of Case 3.2: (S_2, S_3, S_5) (Continued)

Case No.	Condition (S_a, S_b, S_c)	Destination Node out of	t_i	Node Disjoint Path	
3.7.1	$t_\ell \overset{E}{\Rightarrow} S_1^E$ is entirely not used	all	top/left	$t_i \overset{NW}{\Rightarrow} S_1^W \overset{NE}{\Rightarrow} (0,k) \overset{E}{\Rightarrow} (-1,0) \to s$	
		rest	min-weight/top	$t_i \overset{SW}{\Rightarrow} S_1^S \overset{W}{\Rightarrow} (0,1) \to s$	
		rest	max-weight/top	$t_i \overset{E}{\Rightarrow} S_4^W$	To connect one dest. node to S_5 from S_4, rotate; apply Case 5.1 (S_2) from Table 4.8; and rotate back.
		rest	the last one (t_ℓ)	$t_\ell \overset{E}{\Rightarrow} S_4^W$	
3.7.2	$t_\ell \overset{E}{\Rightarrow} t_{max}^W \to t_{max}^{NW} \overset{E}{\Rightarrow} S_1^E$ is entirely not used	all	top/left	$t_i \overset{NW}{\Rightarrow} S_1^W \overset{NE}{\Rightarrow} (0,k) \overset{E}{\Rightarrow} (-1,0) \to s$	
		rest	min-weight/top	$t_i \overset{SW}{\Rightarrow} S_1^S \overset{W}{\Rightarrow} (0,1) \to s$	
		rest	max-weight/top (t_{max})	$t_{max} \overset{E}{\Rightarrow} S_4^W$	To connect one dest. node to S_5 from S_4, rotate; apply Case 5.1 (S_2); and rotate back.
		rest	the last one (t_ℓ)	$t_\ell \overset{E}{\Rightarrow} t_{max}^W \to t_{max}^{NW} \overset{E}{\Rightarrow} S_1^E$	
3.7.3	Otherwise (i.e $t_\ell \overset{E}{\Rightarrow} t_{max}^W \to t_{max}^{SW} \overset{E}{\Rightarrow} S_1^E$ is entirely not used)	all	top/left	$t_i \overset{NW}{\Rightarrow} S_1^W \overset{NE}{\Rightarrow} (0,k) \overset{E}{\Rightarrow} (-1,0) \to s$	
		rest	min-weight/top	$t_i \overset{SW}{\Rightarrow} S_1^S \overset{W}{\Rightarrow} (0,1) \to s$	
		rest	max-weight/top (t_{max})	$t_{max} \overset{E}{\Rightarrow} S_4^W$	To connect one dest. node to S_5 from S_4, rotate; apply Case 5.1 (S_2); and rotate back.
		rest	the last one (t_ℓ)	$t_\ell \overset{E}{\Rightarrow} t_{max}^W \to t_{max}^{SW} \overset{E}{\Rightarrow} S_1^E$	

Table 4.4: Subcases of Case 3.7: (S_3, S_4, S_5)

Case No.	Condition (S_a, S_b, S_c)	Destination Node		Node Disjoint Path
		out of	t_i	
3.9.1	x of the top/right dest. node $\neq 0$	all	top/right	$t_i \overset{NE}{\Rightarrow} S_1^E \overset{NW}{\Rightarrow} (1, k-1) \rightarrow (-k, 0) \overset{E}{\Rightarrow} (-1, 0) \rightarrow s$
		rest	top/left	$t_i \overset{W}{\Rightarrow} S_1^W \overset{NE}{\Rightarrow} S_5^E \overset{NW}{\Rightarrow} (1, -1) \rightarrow s$
		rest	min-weight/top	$t_i \overset{SW}{\Rightarrow} S_1^S \overset{W}{\Rightarrow} (0, 1) \rightarrow s$
		rest	the last one	$t_i \overset{SE}{\Rightarrow} S_6^N \overset{W}{\Rightarrow} (1, 0) \rightarrow s$
3.9.2	x of the top/right dest. node $= 0$	all	top/right	$t_i \overset{NE}{\Rightarrow} S_5^E \overset{NW}{\Rightarrow} (1, -1) \rightarrow s$
		rest	top/right	$t_i \overset{E}{\Rightarrow} S_1^E \overset{NW}{\Rightarrow} (1, k-1) \rightarrow (-k, 0) \overset{E}{\Rightarrow} (-1, 0) \rightarrow s$
		rest	min-weight/top	$t_i \overset{SW}{\Rightarrow} S_1^S \overset{W}{\Rightarrow} (0, 1) \rightarrow s$
		rest	the last one	$t_i \overset{SE}{\Rightarrow} S_6^N \overset{W}{\Rightarrow} (1, 0) \rightarrow s$

Table 4.5: Subcases of Case 3.9: (S_3, S_5, S_6)

1. Find the three sectors (S_a, S_b, S_c) that do not have any destination nodes. If there are more than three sectors of those, arbitrary choose any three sectors.

2. Based on the value of (S_a, S_b, S_c), identify the case number.

3. For this particular case, locate each destination node using column 3 (Destination Node) in the provided order. In other words, apply each rule on the destination nodes that have not been located yet. For example in Case 3.1, we locate the third destination node by applying the rule "min-weight/top" on the rest of destination nodes after locating the first two destination nodes.

4. The corresponding path is given in column 4 (Node Disjoint Path).

For example, consider the example shown in Figure 4.19. In this example, Sectors S_2, S_3, and S_4 do not contain destination nodes (i.e. (S_2, S_3, S_4)). So, the case is Case 3.1. According to Table 4.1, the first destination node is the top/2$^{\text{nd}}$ left destination node in S_1. This node is $(1, 3)$; and the path is $(1, 3) \rightarrow (0, 4) \overset{NE}{\Rightarrow} (0, 5) \overset{E}{\Rightarrow} (-1, 0) \rightarrow s$ as shown in the figure. Next we locate the second destination node. Excluding $(2, 3)$, the second destination node is the top/left destination node in S_1. This node is $(1, 2)$. And the path to reach it is $(2, 3) \overset{W}{\Rightarrow} (-1, 2) \overset{SW}{\Rightarrow}$

$(-1, 1) \rightarrow s$. Similarly we locate the last two destination nodes.

All NDP in Table 4.1 exist because each path goes through an area that does not contain any destination node. This area is enforced by applying the identification rule for each destination node. For example in Figure 4.19, the top/2$^{\text{nd}}$ left destination node in S_1 is $(1, 3)$. It follows that the triangle with vertices $(0, 6)$, $(0, 4)$, and $(2, 4)$ does not contain destination nodes. As shown in the figure, the portion of the path to $(1, 3)$ in S_1 is entirely contained in this triangle. The same idea is applied on all paths.

Node $(0, k)$ (the top node in S_1) is a special node. It is connected to S_2, S_3, and S_5. Because of this, the path to the top destination node is connected to one of these three sectors. Moreover, $(0, k)$ is the only node in S_1 that is connected to S_5; and its neighbor $(1, k-1)$ is the only node in S_1 (other than $(0, k)$) that is connected to S_3. So, if the path $(1, k-1) \rightarrow (0, k)$ is used to go to S_5, the way to S_3 will be blocked. Because of this, the cases when S_3 and S_5 do not contain destination nodes different from other cases. These cases are Case 3.2 (S_2, S_3, S_5), Case 3.7 (S_3, S_4, S_5), and Case 3.9 (S_3, S_5, S_6). In the following, we show the NDP for these cases.

Table 4.4 provides the NDP for Case 3.7 (S_3, S_4, S_5). These NDP connect two destination nodes from S_1 to S_4. Then, one of these destination nodes is connected to S_5. So, we do not need to connect one of the destination nodes in S_1 directly to S_5 through $(0, k)$; which allows us to use $(0, k)$ to connect the top destination node to S_3. Figure 4.20 shows an example. In this example, the top/left destination node is $(1, 3)$; the min-weight/top destination node (out of the rest) is $(1, 1)$; the max-weight/top destination node t_{max} (out of the rest) is $(2, 2)$; and the last destination node t_ℓ is $(1, 2)$. This example does not follow Case 3.7.1 because the

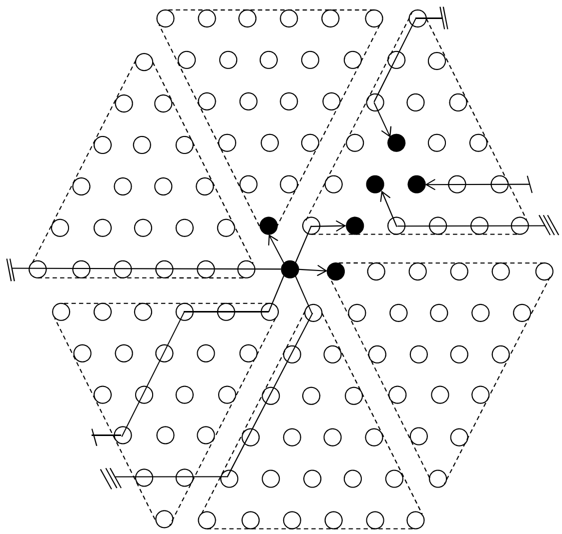


Figure 4.20: Example of Case 3.7.3 (H_6)

path $(1,2) \overset{E}{\Rightarrow} (4,2)$ is blocked by t_{max}. Also, it does not follow Case 3.7.2 because the path $(1,2) \to (1,3) \overset{E}{\Rightarrow} (3,3)$ is blocked by the top/left destination node $(1,3)$. At this point, the path $(1,2) \to (2,1) \overset{E}{\Rightarrow} (5,1)$ must be entirely available because: 1) t_{max} cannot block it, and 2) we assume there are at most two destination nodes have $y = 1$. So, this example follows Case 3.7.3.

Table 4.5 provides the NDP for Case 3.9 (S_3, S_5, S_6). Unlike Case 3.7, we cannot connect a destination node to S_4 because it is possible that this sector contains one or two destination node(s). Instead, we connect two destination nodes to S_3 and S_5 directly from S_1. To prevent the situation of blocking the way to S_3 as explained above, we enforce a condition based on the x-value of the top/right destination node (as shown in Table 4.5). Figure 4.21 shows an example. This example follows Case 3.9.1 because the x-value of the top/right destination node is not equal to zero.

Case 3.2 (S_2, S_3, S_5) is different. To explain how it is different, consider the

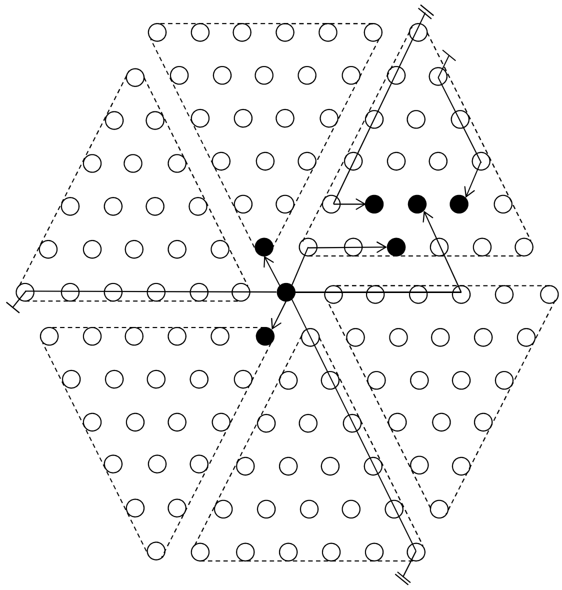


Figure 4.21: Example of Case 3.9.1 (H_6)

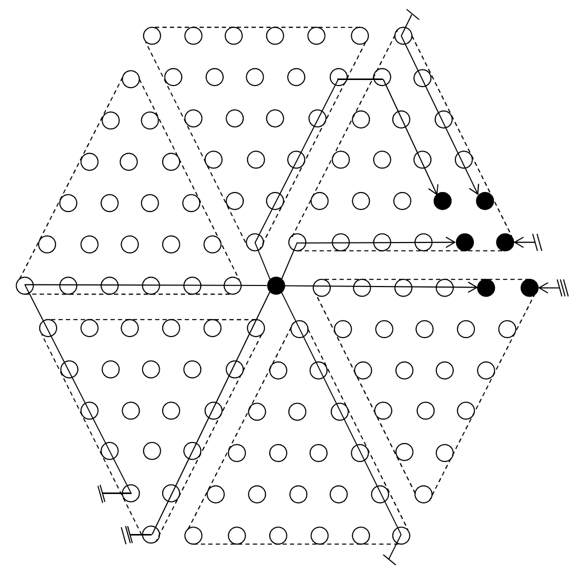


Figure 4.22: Example of Case 3.2.3.3 (H_6)

example in Figure 4.22. In this example, we want to connect three destination nodes to sectors S_2, S_3, and S_5. Unlike all other cases, it is impossible to connect node $(5, 1)$ without going through S_4 or S_6 which contain two destination nodes.

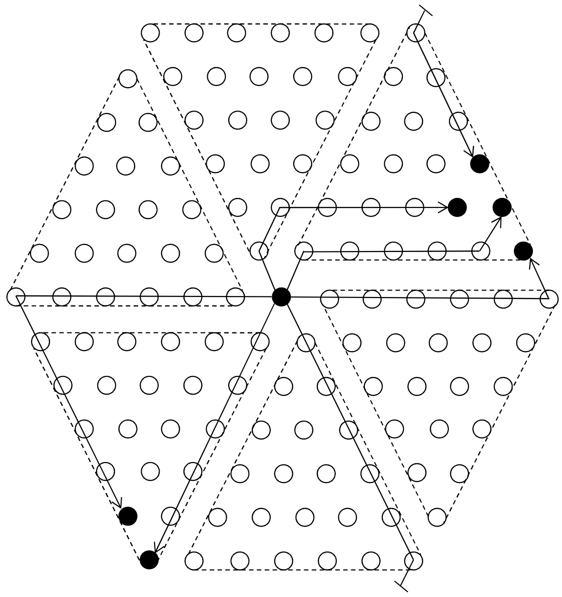


Figure 4.23: Example of Case 3.2.4.1 (H_6)

To solve this problem, we need to take into consideration the locations of the destination nodes in S_4 and S_6. Therefore, we provide the NDP for these destination nodes along with the NDP for the destination nodes in S_1. Table 4.2 provides the NDP for the four destination nodes in S_1 if these paths do not go through S_4 and S_6 (for example, Case 3.2.1). Otherwise, this table provides the NDP for all six destination nodes (for example, Case 3.2.2.2). The example shown in Figure 4.22 follows Case 3.2.3.3 because the east side of S_1 (i.e. S_1^E) has two destination nodes ($(4, 2)$ and $(5, 1)$) and S_4 has no destination nodes. Another example is given in Figure 4.23. This example follows Case 3.2.4.1 and its NDP are given in Table 4.3.

Tables 4.1, 4.2, 4.3, 4.4, and 4.5 cover all possible cases of Case 3 (four destination nodes in S_1). And this concludes this proof. □

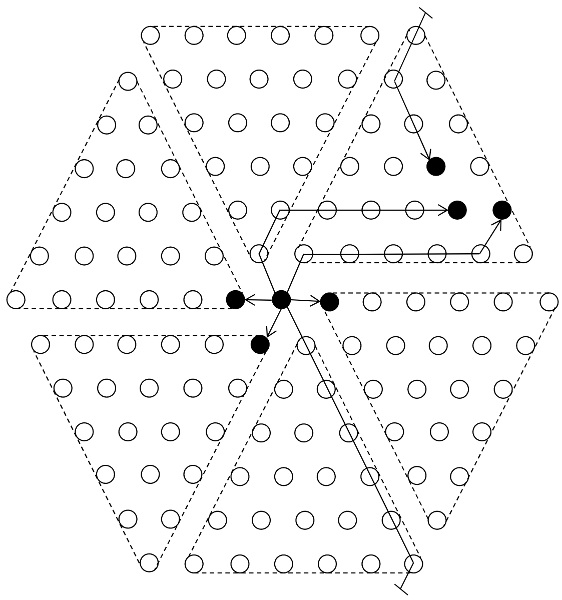


Figure 4.24: Example of Case 4.3 (H_6)

Case 4: Three destination nodes in S_1

In this case, three destination nodes exist in sector S_1. Theorem 4.2.4 shows the NDP for this case.

Theorem 4.2.4. *In a hexagonal mesh network H_k where k is the network diameter, let the source node be $s = (0, 0)$ and the set of destination nodes be $T = \{t_j = (t_{j_x}, t_{j_y}) | 1 \leq j \leq 6\}$ such that three destination nodes exist in S_1. Then, there exist NDP $\mathbb{P}(s, T)$.*

Proof. In this case, at least two sectors (out of $S_2, S_3, \ldots, S_6$) do not contain any destination nodes. Let (S_a, S_b) denote these two sectors where $a, b \in \{2, \ldots, 6\}$. Then, there are 10 cases in the form of (S_a, S_b). Table 4.6 provides the NDP for all 10 cases. Similar to Case 3 (four destination nodes in S_1), we construct the NDP for Case 4 taking into consideration the following:

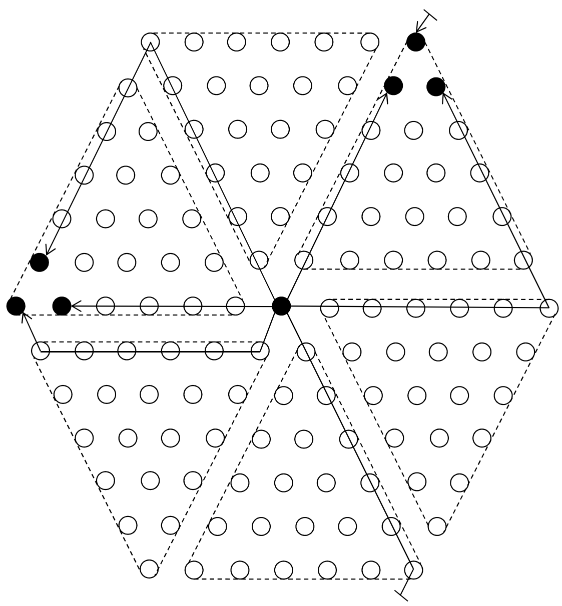


Figure 4.25: Example of Case 4.9.1 (H_6)

1. All paths go through areas that do not contain any destination nodes by applying the rules given in column Destination Node. For example in Figure 4.24 which follows Case 4.3 (S_2, S_5), the path to the destination node $(2, 3)$ goes through S_5 and the triangle with vertices $(0, 6)$, $(0, 4)$, and $(2, 4)$. Sector S_5 does not contain destination nodes because the case is (S_2, S_5); and the triangle area does not contain a destination node because $(2, 3)$ is the top destination node in S_1.

2. The top destination node is always connected to S_3, S_5, or S_2 because the top node $(0, k)$ is connected to these sectors only.

3. The NDP connect two destination nodes from S_1 to the sectors specified by the case identifier (S_a, S_b) without crossing other sectors except in Case 4.9 (S_4, S_6). In this case, we provide the NDP for all six destination nodes. Table 4.7 provides the NDP for Case 4.9 (S_4, S_6). This case is special because

Case No.	Conditions (AND)			Destination Node		Node Disjoint Path
	1 (S_a, S_b)	2	3	out of	t_i	
4.1	(S_2, S_3)	at most one dest. node has $y = 1$		all	top/2nd left	$t_i \overset{NW}{\Rightarrow} S_1^W \overset{NE}{\Rightarrow} (0,k) \overset{E}{\Rightarrow} (-1,0) \rightarrow s$
				rest	top/left	$t_i \overset{W}{\Rightarrow} S_2^E \overset{SW}{\Rightarrow} (-1,1) \rightarrow s$
				rest	the last one	$t_i \overset{SW}{\Rightarrow} S_1^S \overset{W}{\Rightarrow} (0,1) \rightarrow s$
4.2	(S_2, S_4)	at most two dest. nodes have $y = 1$		all	top/left	$t_i \overset{W}{\Rightarrow} S_2^E \overset{SW}{\Rightarrow} (-1,1) \rightarrow s$
				rest	min-weight/top	$t_i \overset{SW}{\Rightarrow} S_1^S \overset{W}{\Rightarrow} (0,1) \rightarrow s$
				rest	the last one	$t_i \overset{E}{\Rightarrow} S_4^W \overset{NE}{\Rightarrow} S_4^N \overset{E}{\Rightarrow} (0,-1) \rightarrow s$
4.3	(S_2, S_5)	at most one dest. node has $y = 1$		all	top/2nd left	$t_i \overset{NW}{\Rightarrow} S_1^W \overset{NE}{\Rightarrow} S_5^E \overset{NW}{\Rightarrow} (1,-1) \rightarrow s$
				rest	top/left	$t_i \overset{W}{\Rightarrow} S_2^E \overset{SW}{\Rightarrow} (-1,1) \rightarrow s$
				rest	the last one	$t_i \overset{SW}{\Rightarrow} S_1^S \overset{W}{\Rightarrow} (0,1) \rightarrow s$
4.4	(S_2, S_6)	at most two dest. nodes have $y = 1$		all	top/left	$t_i \overset{W}{\Rightarrow} S_2^E \overset{SW}{\Rightarrow} (-1,1) \rightarrow s$
				rest	min-weight/top	$t_i \overset{SW}{\Rightarrow} S_1^S \overset{W}{\Rightarrow} (0,1) \rightarrow s$
				rest	the last one	$t_i \overset{SE}{\Rightarrow} S_6^N \overset{W}{\Rightarrow} (1,0) \rightarrow s$
4.5	(S_3, S_4)			all	top/2nd left	$t_i \overset{NW}{\Rightarrow} S_1^W \overset{NE}{\Rightarrow} (0,k) \overset{E}{\Rightarrow} (-1,0) \rightarrow s$
				rest	min-weight/top	$t_i \overset{SW}{\Rightarrow} S_1^S \overset{W}{\Rightarrow} (0,1) \rightarrow s$
				rest	the last one	$t_i \overset{E}{\Rightarrow} S_4^W \overset{NE}{\Rightarrow} S_4^N \overset{E}{\Rightarrow} (0,-1) \rightarrow s$
4.6.1	(S_3, S_5)	at most one dest. node has $y = 1$	x of the top/right dest. node $\neq 0$	all	top/right	$t_i \overset{NE}{\Rightarrow} S_1^E \overset{NW}{\Rightarrow} (1,k-1) \rightarrow (-k,0) \overset{E}{\Rightarrow} (-1,0) \rightarrow s$
				rest	top/left	$t_i \overset{W}{\Rightarrow} S_1^W \overset{NE}{\Rightarrow} S_5^E \overset{NW}{\Rightarrow} (1,-1) \rightarrow s$
				rest	the last one	$t_i \overset{SW}{\Rightarrow} S_1^S \overset{W}{\Rightarrow} (0,1) \rightarrow s$
4.6.2		at most two dest. nodes have $y = 1$	x of the top/right dest. node $= 0$	all	top/right	$t_i \overset{NE}{\Rightarrow} S_5^E \overset{NW}{\Rightarrow} (1,-1) \rightarrow s$
				rest	top/right	$t_i \overset{E}{\Rightarrow} S_1^E \overset{NW}{\Rightarrow} (1,k-1) \rightarrow (-k,0) \overset{E}{\Rightarrow} (-1,0) \rightarrow s$
				rest	the last one	$t_i \overset{SW}{\Rightarrow} S_1^S \overset{W}{\Rightarrow} (0,1) \rightarrow s$
4.7	(S_3, S_6)			all	top/2nd left	$t_i \overset{NW}{\Rightarrow} S_1^W \overset{NE}{\Rightarrow} (0,k) \overset{E}{\Rightarrow} (-1,0) \rightarrow s$
				rest	min-weight/top	$t_i \overset{SW}{\Rightarrow} S_1^S \overset{W}{\Rightarrow} (0,1) \rightarrow s$
				rest	the last one	$t_i \overset{SE}{\Rightarrow} S_6^N \overset{W}{\Rightarrow} (1,0) \rightarrow s$
4.8	(S_4, S_5)			all	top/2nd left	$t_i \overset{NW}{\Rightarrow} S_1^W \overset{NE}{\Rightarrow} S_5^E \overset{NW}{\Rightarrow} (1,-1) \rightarrow s$
				rest	min-weight/top	$t_i \overset{SW}{\Rightarrow} S_1^S \overset{W}{\Rightarrow} (0,1) \rightarrow s$
				rest	the last one	$t_i \overset{E}{\Rightarrow} S_4^W \overset{NE}{\Rightarrow} S_4^N \overset{E}{\Rightarrow} (0,-1) \rightarrow s$
4.9	(S_4, S_6)			Look at Table 4.7		
4.10	(S_5, S_6)			all	top/2nd left	$t_i \overset{NW}{\Rightarrow} S_1^W \overset{NE}{\Rightarrow} S_5^E \overset{NW}{\Rightarrow} (1,-1) \rightarrow s$
				rest	min-weight/top	$t_i \overset{SW}{\Rightarrow} S_1^S \overset{W}{\Rightarrow} (0,1) \rightarrow s$
				rest	the last one	$t_i \overset{SE}{\Rightarrow} S_6^N \overset{W}{\Rightarrow} (1,0) \rightarrow s$

Table 4.6: Subcases of Case 4: three destination nodes in S_1

it is impossible to connect the top destination node to either S_4 or S_6. Instead, we connect this destination node to either S_3, S_5, or S_2 based on the locations of the destination nodes that are not in S_1. Figure 4.25 shows an example of Case 4.9.1.

Tables 4.6 and 4.7 cover all possible cases of Case 4 (three destination nodes in S_1). And this concludes this proof.

□

Case No.	Conditions (AND)		Destination Node		Node Disjoint Path
	1	2	out of	t_i	
4.9.1	S_5 does not contain dest. nodes		all		To connect two dest. nodes from S_1 to S_5 and S_6, apply Case 4.10 (S_5, S_6) from Table 4.6 on S_1.
				the ones in S_2 and S_3	To distribute the remaining destination nodes in S_2 and S_3 over S_2, S_3, and S_4, rotate; apply the appropriate case from either Table 4.6 or Table 4.8; and, rotate back. For example, if S_3 contains three dest. nodes, rotate; apply Case 4.4 (S_2, S_6); and rotate back.
4.9.2	S_5 contains one dest. node		all	top/2nd left	If $(k, -k)$ is available, the path is $t_i \overset{NW}{\Rightarrow} S_1^W \overset{NE}{\Rightarrow} (0, k) \rightarrow (k, -k)$. Otherwise, the path is $t_i \overset{NW}{\Rightarrow} S_1^W \overset{NE}{\Rightarrow} (0, k) \rightarrow (k-1, -k)$. Then, to connect one dest. node from S_5 to S_6, rotate; apply Case 5.1 (S_2) on the node at the end of this path and the dest. node in S_5; and rotate back.
			rest	min-weight/top	$t_i \overset{SW}{\Rightarrow} S_1^S \overset{W}{\Rightarrow} (0, 1) \rightarrow s$
			rest	the last one	$t_i \overset{E}{\Rightarrow} S_4^W \overset{NE}{\Rightarrow} S_4^N \overset{E}{\Rightarrow} (0, -1) \rightarrow s$
				the ones in S_2 and S_3	If both in S_2, rotate; apply Case 5.1 (S_2); and rotate back. If both in S_3, rotate; apply Case 5.5 (S_6); and rotate back.
4.9.3.1	S_5 contains two dest. nodes	S_2 contains a dest. node		the ones in S_1	To connect two dest. nodes to S_4 and S_5, apply Case 4.5 (S_3, S_4).
				the ones in S_5	To connect one dest. node to S_6, rotate; apply Case 5.1 (S_2); and rotate back.
4.9.3.2		S_3 has a dest. node		the ones in S_1	To connect two dest. nodes to S_4 and S_5, apply Case 4.2 (S_2, S_4).
				the ones in S_5	To connect one dest. node to S_6, rotate; apply Case 5.1 $((S_2))$; and rotate back.
4.9.4	S_5 contains three dest. nodes			the ones in S_1	To connect two dest. nodes to S_2 and S_3, apply Case 4.1 (S_2, S_3).
				the ones in S_5	To connect two dest. nodes to S_4 and S_6, rotate; apply Case 4.4 (S_2, S_6); and rotate back.

Table 4.7: Subcases of Case 4.9: (S_4, S_6)

Case 5: Two destination nodes in S_1

In this case, two destination nodes exist in sector S_1. Theorem 4.2.5 shows the NDP for this case.

Theorem 4.2.5. *In a hexagonal mesh network H_k where k is the network diameter, let the source node be $s = (0, 0)$ and the set of destination nodes be $T = \{t_j = (t_{j_x}, t_{j_y}) | 1 \leq j \leq 6\}$ such that two destination nodes exist in S_1. Then, there exist NDP $\mathbb{P}(s, T)$.*

Proof. In this case, four destination nodes do not exist in S_1. It follows that at least one sector (out of $S_2, S_3, \ldots, S_6$) does not contain any destination nodes. Let (S_a) denote this sector where $a \in \{2, \ldots, 6\}$ (not to be confused with the sector number S_i). Then, there are 5 cases in the form of (S_a). Table 4.8 provides the

Case No.	Conditions (AND) 1 (S_a)	2	Destination Node out of	t_i	Node Disjoint Path
5.1	(S_2)		all	top/2[nd] left	$t_i \overset{NW}{\Rightarrow} S_1^W \rightarrow S_2^E \overset{SW}{\Rightarrow} (-1,1) \rightarrow s$
			rest	the last one	$t_i \overset{SW}{\Rightarrow} S_1^S \overset{W}{\Rightarrow} (0,1) \rightarrow s$
5.2	(S_3)		all	top/2[nd] left	$t_i \overset{NW}{\Rightarrow} S_1^W \overset{NE}{\Rightarrow} (0,k) \overset{E}{\Rightarrow} (-1,0) \rightarrow s$
			rest	the last one	$t_i \overset{SW}{\Rightarrow} S_1^S \overset{W}{\Rightarrow} (0,1) \rightarrow s$
5.3.1	(S_4)	$(0,k)$ is a dest. node and $(1,k-1)$ is not a dest. node	all	top	$t_i \rightarrow (1,k-1) \overset{E}{\Rightarrow} (0,-1) \rightarrow s$
			rest	the last one	$t_i \overset{SW}{\Rightarrow} S_1^S \overset{W}{\Rightarrow} (0,1) \rightarrow s$
5.3.2		otherwise	all	min-weight/top	$t_i \overset{SW}{\Rightarrow} S_1^S \overset{W}{\Rightarrow} (0,1) \rightarrow s$
			rest	the last one	$t_i \overset{E}{\Rightarrow} S_4^W \overset{NE}{\Rightarrow} S_4^N \overset{E}{\Rightarrow} (0,-1) \rightarrow s$
5.4	(S_5)		all	top/2[nd] left	$t_i \overset{NW}{\Rightarrow} S_1^W \overset{NE}{\Rightarrow} S_5^E \overset{NW}{\Rightarrow} (1,-1) \rightarrow s$
			rest	the last one	$t_i \overset{SW}{\Rightarrow} S_1^S \overset{W}{\Rightarrow} (0,1) \rightarrow s$
5.5	(S_6)		all	min-weight/top	$t_i \overset{SW}{\Rightarrow} S_1^S \overset{W}{\Rightarrow} (0,1) \rightarrow s$
			rest	the last one	$t_i \overset{W}{\Rightarrow} S_2^E \overset{SW}{\Rightarrow} (-1,1) \rightarrow s$

Table 4.8: Subcases of Case 5: two destination nodes in S_1

NDP for all 5 cases. □

Case 6: One destination node in S_1

If S_1 contains only one destination node t_i, the path is $t_i \overset{SW}{\Rightarrow} S_1^S \overset{W}{\Rightarrow} (0,1) \rightarrow s$.

4.2.1 One-to-Many Node Disjoint Paths Routing Algorithm

In the previous section, we provide the NDP for those destination nodes which are in Sector 1 (i.e. S_1). Now, if some nodes are in sector, say S_i, $i \neq 1$, then the

Algorithm 4 One-to-Many NDP Routing in Hexagonal Mesh Network H_k

Input: H_k, $T = \{t_j = (t_{j_x}, t_{j_y}) | 1 \leq j \leq 6\}$, $s = (0, 0) \notin T$
Output: $\mathbb{P}(s, T)$

procedure $OneToMany\_NDP(H_k,T,s)$
  for $1 \leq i \leq 6$ **do**
    switch number of un-reached dest. nodes in S_1
      $6 : \mathbb{P}(s, T) = Case1(G_k, T, s)$;
      $5 : \mathbb{P}(s, T) = Case2(G_k, T, s)$;
      $4 : \mathbb{P}(s, T) = Case3(G_k, T, s)$;
      $3 : \mathbb{P}(s, T) = Case4(G_k, T, s)$;
      $2 : \mathbb{P}(s, T) = Case5(G_k, T, s)$;
      $1 : \mathbb{P}(s, T) = Case6(G_k, T, s)$;
      0 : Do nothing;
    end switch
    $G_k = G_k * \rho$; ▷ Rotate in counter-clock direction
    $\mathbb{P}(s, T) = \mathbb{P}(s, T) * \rho$;
  end for
  return $\mathbb{P}(s, T)$;
end procedure

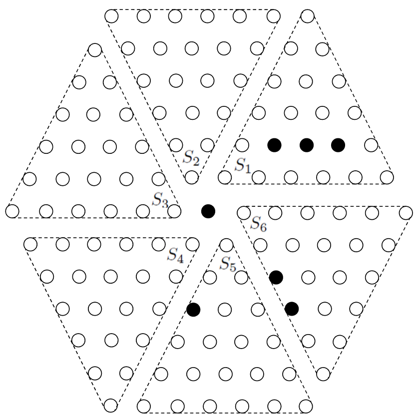
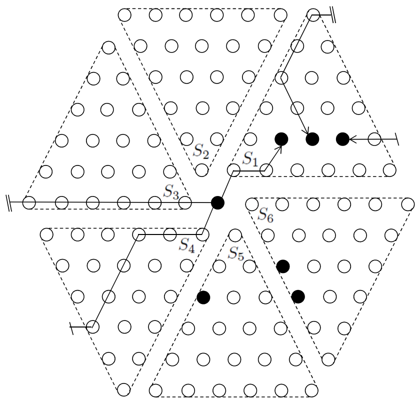


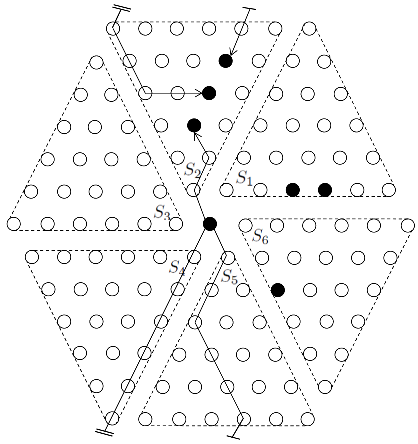
Figure 4.26: Example of Algorithm 4: Initial network

network can be rotated such that these nodes become in Sector 1. Then, we apply the above given NDP algorithm to reach these nodes. After this, the network is rotated such that these nodes belong to the original location. (see Algorithm 4.)

For example, consider the network provided in Figure 4.26 as an input to the

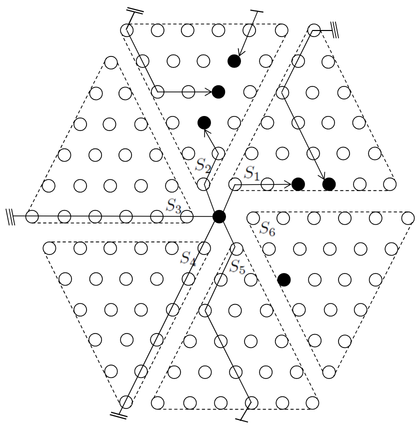


(a) NDP

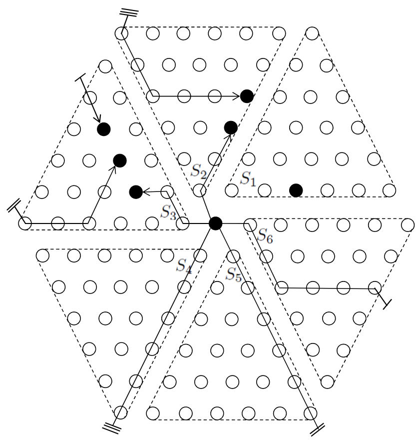


(b) Rotation

Figure 4.27: Example of Algorithm 4: 1st Iteration



(a) NDP



(b) Rotation

Figure 4.28: Example of Algorithm 4: 2nd Iteration

algorithm. In this instance, S_1, S_5, and S_6 contain three, one, and two destination nodes, respectively. In the first iteration, the algorithm constructs the NDP to the three destination nodes in S_1 by applying one of the following cases arbitrary: Case 4.1 (S_2, S_3), Case 4.2 (S_2, S_4), or 4.5 (S_3, S_4) from Table 4.6. These are the cases because S_1 contains three destination nodes and S_2, S_3, and S_4 do not contain destination or used nodes. Assuming the algorithm chooses Case 4.5 (S_3, S_4), the

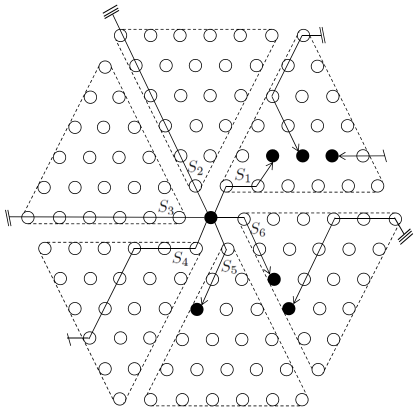


Figure 4.29: Example of Algorithm 4: 6^{th} Iteration

resultant NDP are shown in Figure 4.27a. Then, the network and those constructed NDP are rotated as shown in Figure 4.27b. As a result of this rotation, S_6 becomes S_1; and now S_1 contains two destination nodes. These destination nodes will be reached in the next iteration.

In the second iteration, the algorithm applies Case 5.2 (S_3) from Table 4.8 since S_3 is the only sector that does not contain destination or used nodes. The result is shown in Figure 4.28a. Then, the network is rotated to get the updated network shown in Figure 4.28b.

After the final rotation in the sixth iteration, the network is returned to its initial location and all destination nodes have been reached. The final result is shown in Figure 4.29.

4.3 Conclusion

In this chapter we provide and prove an algorithm to construct all NDP from a single source node to a set of destination nodes in Hexagonal Mesh Networks (HMNs). This algorithm constructs six NDP and this is the maximum number of

NDP that can be obtained because the degree of the nodes is six.

Chapter 5: Conclusion

Achieving high computing performance in parallel computing systems critically depends on finding a set of mutually node disjoint paths (NDP). In this work we provide and prove some novel algorithms to find a set of the maximum number of one-to-many NDP from a source node to a set of destination nodes in Generalized Hypercube (GH), dense Gaussian, and Hexagonal Mesh networks.

5.1 Findings

In Chapter 2, the findings are:

1. Proposing an algorithm to solve the one-to-many NDP routing problem for two-dimensional GH,
2. Proposing another algorithm to solve the same problem for n-dimensional GH,
3. Theoretically proving that both algorithms always return a solution,
4. Theoretically proving that the length of the path from s to t_i returned by the algorithms is bounded between the shortest distance and $2n-1$, where n is the dimension of the GH,
5. Showing that the time complexity of the algorithm is $O({k_{max}}^2 n^3)$ where $k_{max} = \max_{0 \leq i \leq (n-1)} \{k_i\}$ and k_i is the number of nodes in dimension i, and

6. Simulation results showing that the longest path lengths are close to the shortest distance plus one, which is less than the theoretical upper bound for $n > 2$.

In Chapter 3, the findings are:

1. Proposing an efficient algorithm to solve the one-to-many NDP routing problem in dense Gaussian networks (DGNs) without depending on the network size,

2. Theoretically proving that the proposed algorithm always returns a solution,

3. Theoretically proving that the sum of NDP lengths from the source node to the destination nodes constructed by the proposed algorithm is bounded between the sum of the shortest paths and this sum plus $(6k - 11)$ where k is the diameter,

4. Analyzing the time complexity to show that the time complexity of the algorithm is constant $O(1)$, and

5. The algorithm executing results show that on the average the sum of NDP lengths is only about 10% more than the sum of the shortest paths.

In Chapter 4, the findings are:

1. Proposing an efficient algorithm to solve the one-to-many NDP routing problem in dense Hexagonal Mesh networks (HMNs) without depending on the network size, and

2. Theoretically proving that the proposed algorithm always returns a solution.

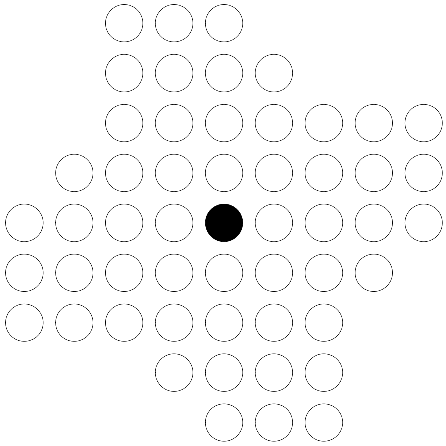


Figure 5.1: Gaussian network ($\alpha = 2 + 7\mathbf{i}$)

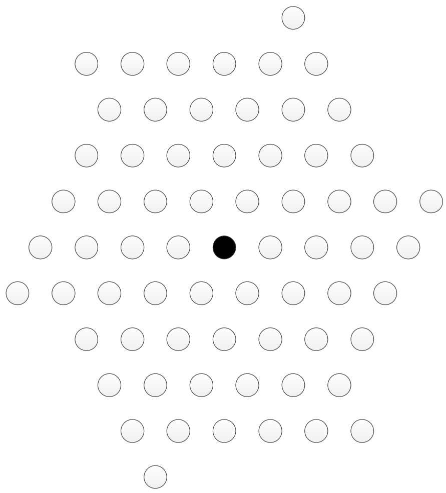


Figure 5.2: EJ network ($\alpha = 2 + 7\rho$)

5.2 Future Work

In the following, we list some possible future research directions:

1. **One-to-Many Node Disjoint Paths:**

   In this work, we assume the generators of dense Gaussian and Hexagonal Mesh networks are $\alpha = k + (k+1)\mathbf{i}$ and $\alpha = (k+1) + k\rho$ where k is the diameter, respectively. These networks belong to the general classes:

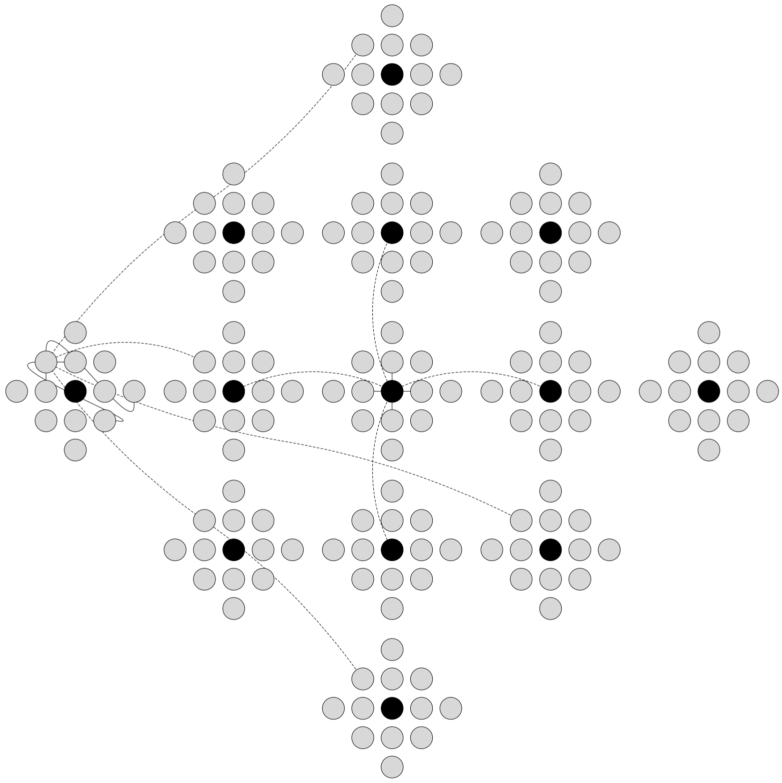


Figure 5.3: Higher-dimensional Gaussian network

Gaussian and EJ networks. The general Gaussian networks are generated by $\alpha = a + b\mathbf{i}$ where $a, b \in \mathbb{Z}$; the general EJ networks are generated by $\alpha = a + b\rho$ where $a, b \in \mathbb{Z}$. Figure 5.1 shows an example of Gaussian network generated by $\alpha = 2 + 7\mathbf{i}$; and Figure 5.2 shows an example of EJ network generated by $\alpha = 2 + 7\rho$. Generalizing the NDP routing algorithms given in this work to cover the general class of Gaussian and EJ networks is one of the possible future research directions.

Another research direction is finding the shortest NDP as the algorithms provided in this work do not find the shortest NDP.

Recently, Shamaei, Bose, and Flahive introduced higher dimensional Gaus-

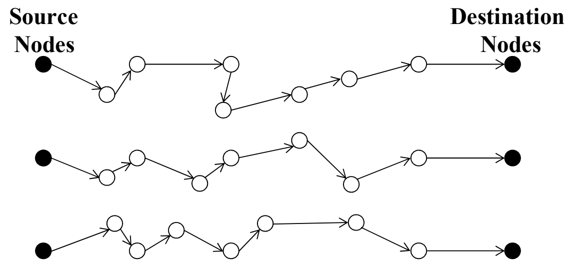


Figure 5.4: Many-to-many k-disjoint path cover

sian networks [36]. These networks support dimensions more than two. Figure 5.3 shows a Gaussian network, where each node is represented by two Gaussian integers. Generalizing the one-to-many NDP routing algorithm proposed in Chapter 3 for the basic dense Gaussian network to support higher dimensional Gaussian network is one of the possible future research directions.

2. **Paired many-to-many k-disjoint path cover:**

   A set of many-to-many k-disjoint path cover connects a set of source nodes with a set of destination nodes using NDP that cover all nodes in the network (see Figure 5.4). This kind of NDP is useful for the applications that required full utilization of the nodes. Chen solved this problem for Hypercube [11]. Finding these NDP in Generalized Hypercube, dense Gaussian, or Hexagonal Mesh networks is not solved yet. Moreover, Paired one-to-one and one-to-many k-disjoint path cover in these networks are not solved yet. These are some of the open problems that need further investigation.

3. **Paired many-to-many k-disjoint path cover in fault network:**

   This problem is the same as the previous one except that some nodes are

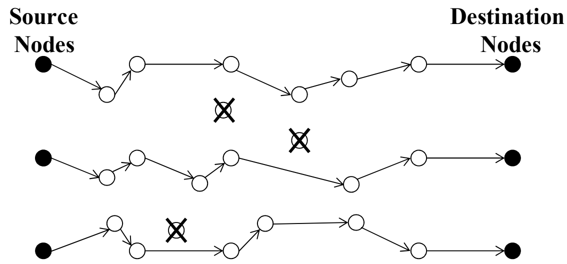


Figure 5.5: Many-to-many k-disjoint path cover in fault network

faulty and cannot be used by any node disjoint path (see Figure 5.5). This problem is important because when the number of nodes increases, the probability of some node failure will increase. Avoiding these faulty nodes improves the computing performance.

Bibliography

[1] Nibdita Adhikari and CR Tripathy. Folded dualcube: A new interconnection topology for parallel systems. In *International Conference on Information Technology*, pages 75–78, Chengdu, 21-24 April 2008. IEEE Computer Society Press, Washington, DC.

[2] Narasimha R Adiga et al. An overview of the BlueGene/L supercomputer. In *ACM/IEEE 2002 Supercomputing Conference*, pages 60–60, Baltimore, Maryland, 16-22 November 2002. IEEE Computer Society Press, Washington, DC.

[3] Sheldon B Akers and Balakrishnan Krishnamurthy. A group-theoretic model for symmetric interconnection networks. *IEEE Transactions on Computers*, 38(4):555–566, 1989.

[4] Bader Albader, Bella Bose, and Mary Flahive. Efficient communication algorithms in hexagonal mesh interconnection networks. *Parallel and Distributed Systems, IEEE Transactions on*, 23(1):69–77, 2012.

[5] L.N. Bhuyan and D.P. Agrawal. Generalized hypercube and hyperbus structures for a computer network. *IEEE Transactions on Computers*, C-33(4):323 –333, Apr. 1984.

[6] Arthur S Bland, Ricky A Kendall, Douglas B Kothe, James H Rogers, and Galen M Shipman. Jaguar: The worlds most powerful computer. *Memory (TB)*, 300(62):362, 2009.

[7] Bella Bose, Bob Broeg, Younggeun Kwon, and Yaagoub Ashir. Lee distance and topological properties of k-ary n-cubes. *IEEE Transactions on Computers*, 44(8):1021–1030, 1995.

[8] Antoine Bossard and Keiichi Kaneko. The set-to-set disjoint-path problem in perfect hierarchical hypercubes. *The Computer Journal*, 55(6):769–775, 2012.

[9] Antoine Bossard, Keiichi Kaneko, and Shietung Peng. Node-to-set disjoint-path routing in metacube. In *International Conference on Parallel and Distributed Computing, Applications and Technologies*, pages 57–62, Hiroshima, 8-11 December 2009. IEEE Computer Society Press, Washington, DC.

[10] Antoine Bossard, Keiichi Kaneko, and Shietung Peng. A new node-to-set disjoint-path algorithm in perfect hierarchical hypercubes. *The Computer Journal*, 54(8):1372–1381, 2011.

[11] Xie-Bin Chen. Paired many-to-many disjoint path covers of the hypercubes. *Information Sciences*, 236(0):218 – 223, 2013.

[12] P.F. Corbett. Rotator graphs: An efficient topology for point-to-point multiprocessor networks. *IEEE Transactions on Parallel and Distributed Systems*, 3(5):622 –626, Sep. 1992.

[13] William James Dally and Brian Patrick Towles. *Principles and Practices of Interconnection Networks*. Morgan Kaufmann, Burlington, Massachusetts, 2004.

[14] Jose Duato, Sudhakar Yalamanchili, and Lionel Ni. *Interconnection Networks*. Morgan Kaufmann, Burlington, Massachusetts, 2003.

[15] M. Flahive and B. Bose. The topology of Gaussian and Eisenstein-Jacobi interconnection networks. *IEEE Transactions on Parallel and Distributed Systems*, 21(8):1132 –1142, Aug. 2010.

[16] Qian-Ping Gu and Shietung Peng. Fault tolerant routing in toroidal networks. In *First Aizu International Symposium on Parallel Algorithms/Architecture Synthesis*, pages 162–168, Fukushima, 15-17 March 1995. IEEE Computer Society Press, Washington, DC.

[17] G Godfrey Harold Hardy and Edward M Wright. *An Introduction to the Theory of Numbers*. Oxford University Press, Oxford, 1979.

[18] Zaid Hussain. *On shortest disjoint paths and hamiltonian cycles in some interconnection networks*. PhD thesis, Oregon State University, Corvallis, 2011.

[19] JH Jordan and CJ Potratz. Complete residue systems in the Gaussian integers. *Mathematics Magazine*, 38(1):1–12, 1965.

[20] Keiichi Kaneko and Shietung Peng. Node-to-set disjoint paths routing in dual-cube. In *International Symposium on Parallel Architectures, Algorithms, and Networks*, pages 77–82, Sydney, 7-9 May 2008. IEEE Computer Society Press, Washington, DC.

[21] Keiichi Kaneko and Shietung Peng. Set-to-set disjoint paths routing in dual-cubes. In *Parallel and Distributed Computing, Applications and Technologies, 2008. PDCAT 2008. Ninth International Conference on*, pages 129–136, 2008.

[22] Keiichi Kaneko and Yasuto Suzuki. An algorithm for node-to-set disjoint paths problem in rotator graphs. *IEICE Transaction on Information and Systems*, 84(9):1155–1163, 2001.

[23] Keiichi Kaneko and Yasuto Suzuki. Node-to-set disjoint paths problem in pancake graphs. *IEICE Transaction on Information and Systems*, 86(9):1628–1633, 2003.

[24] Vipin Kumar, Ananth Grama, Anshul Gupta, and George Karypis. *Introduction to Parallel Computing*, volume 110. Benjamin Cummings, Redwood City, 1994.

[25] Cheng-Nan Lai. Optimal construction of all shortest node-disjoint paths in hypercubes with applications. *IEEE Transactions on Parallel and Distributed Systems*, 23(6):1129–1134, 2012.

[26] Cheng-Nan Lai, Gen-Huey Chen, and Dyi-Rong Duh. Constructing one-to-many disjoint paths in folded hypercubes. *IEEE Transactions on Computers*, 51(1):33–45, Jan. 2002.

[27] Yamin Li and Shietung Peng. Dual-cubes: A new interconnection network for high-performance computer clusters. In *Proceedings of the International Computer Architecture*, pages 51–57, Vancouver, British Columbia, 10-14 June 2000. IEEE Computer Society Press, Washington, DC.

[28] Shan Ling and Weidong Chen. Node-to-set disjoint paths in biswapped networks. *The Computer Journal*, 2013.

[29] László Lipták, Eddie Cheng, Jong-Seok Kim, and Sung Won Kim. One-to-many node-disjoint paths of hyper-star networks. *Discrete Applied Mathematics*, 160(13):2006–2014, 2012.

[30] Di Liu and Jing Li. Many-to-many n-disjoint path covers in n-dimensional hypercubes. *Inf. Process. Lett.*, 110(0):580–584, July 2010.

[31] C. Martínez, R. Beivide, E. Stafford, M. Moreto, and E.M. Gabidulin. Modeling toroidal networks with the Gaussian integers. *IEEE Transactions on Computers*, 57(8):1046 –1056, Aug. 2008.

[32] C. Martínez, E. Stafford, R. Beivide, and E.M. Gabidulin. Modeling hexagonal constellations with Eisenstein-Jacobi graphs. *Problems of Information Transmission*, 44(1):1–11, 2008.

[33] Carmen Martínez, Enrique Vallejo, Ramón Beivide, Cruz Izu, and Miquel Moretó. Dense Gaussian networks: Suitable topologies for on-chip multiprocessors. *International Journal of Parallel Programming*, 34(3):193–211, 2006.

[34] Behrooz Parhami. *Introduction to Parallel Processing: Algorithms and Architectures*. Springer, New York, 1999.

[35] Shietung Peng and Keiichi Kaneko. Set-to-set disjoint-path routing in metacube. In *Parallel and Distributed Computing, Applications and Technologies (PDCAT), 2010 International Conference on*, pages 128–137, 2010.

[36] Arash Shamaei, Bella Bose, and Mary Flahive. Higher dimensional Gaussian networks. In *The 28th IEEE International Parallel and Distributed Processing Symposium [submitted]*, 2014.

[37] Yuan-Kang Shih and Shin-Shin Kao. One-to-one disjoint path covers on k-ary n-cubes. *Theoretical Computer Science*, 412(35):4513–4530, 2011.

[38] O. Sinanoglu, M.H. Karaata, and B. AlBdaiwi. An inherently stabilizing algorithm for node-to-node routing over all shortest node-disjoint paths in hypercube networks. *IEEE Transactions on Computers*, 59(7):995–999, 2010.

[39] R. Wu, G. Chen, Y. Kuo, and G. Chang. Node-disjoint paths in hierarchical hypercube networks. *Information Sciences*, 177(19):4200 – 4207, 2007.

[40] Yonghong Xiang and Iain A Stewart. One-to-many node-disjoint paths in (n, k)-star graphs. *Discrete Applied Mathematics*, 158(1):62–70, 2010.

[41] J. Xu. *Topological Structure and Analysis of Interconnection Networks*. Kluwer Academic, Dordrecht, 2001.

[42] Z. Zhang, Z. Guo, and Y. Yang. Efficient all-to-all broadcast in Gaussian on-chip-networks. *IEEE Transactions on Computers*, PP(99):1, 2012.

[43] Sotirios G Ziavras and Sanjay Krishnamurthy. Evaluating the communications capabilities of the generalized hypercube interconnection network. *Concurrency Practice and Experience*, 11(6):281–300, 1999.