PROBLEM SOLVING IN ALGORITHMS

A RESEARCH APPROACH

Sanpawat Kantabutra
Faculty of Engineering, CMU

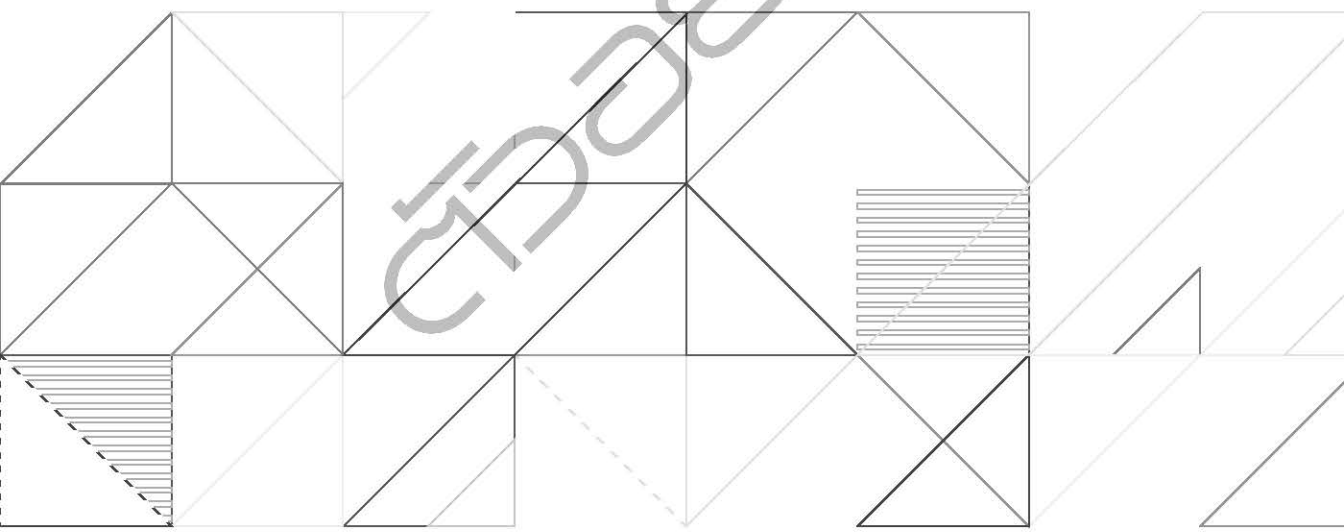
CURRICULUM VITAE

Sanpawat Kantabutra, PhD

Sanpawat Kantabutra is currently Associate Professor in the Theory of Computation Group in the Faculty of Engineering in Chiang Mai University, Thailand. He is also a Thailand Research Fund Career Award researcher and a researcher in the Research Center for Quantum Technology in Chiang Mai University. He earned his PhD in theoretical computer science from Tufts University in Boston in the United States and has regularly published his research findings in some of the world's top journals in theoretical computer science. His current research interests are in approximation and randomized algorithms and quantum computation.

PROBLEM SOLVING IN ALGORITHMS

A RESEARCH APPROACH



Sanpawat Kantabutra
Faculty of Engineering, CMU

Problem Solving in Algorithms

A Research Approach

Editor: Kanchana Kanchanasut
ISBN (e-Book) : 978-616-398-549-1
Author: Sanpawat Kantabutra
Published: Chiang Mai University Press
Office of Research Administration Center
Tel: +66 (0) 5394 3603-4
Fax: +66 (0) 5394 3600
https://cmupress.cmu.ac.th
E-mail: cmupress.th@gmail.com

Printed: March 2021
Price: 315 Baht

Cover design: Somsib Craft

Contact: Chiang Mai University Press
Tel: +66 (0) 53 94 3605 Fax: +66 (0) 53 94 3600
https://cmupress.cmu.ac.th, E-mail: cmupress.th@gmail.com

Appreciation

This book aims to develop critical thinking skills through problem solving. Each chapter is based on published research by the author and demonstrates a detailed thinking process for a specific problem that may arise in a diverse selection of research areas such as communication, biology, data mining, computer architecture and general computer science. The author begins each chapter by describing a single problem of interest and motivations for this study before diving into formal treatments of the topic and ending with comments, suggestions and open problems for further research in the conclusion. The book is modular with self-contained chapters suitable as a reference book for researchers or as a textbook for graduate students in algorithms.

The author emphasizes firm theoretical understanding of each problem with related complexities. Formal treatment of cyber security is an example of such an approach where the problem is properly defined with two computing problems from the economic perspectives of the attackers and defenders. The author first defines a cyber security model, discusses two computing solutions and analyses their complexities that turn out to be NP-complete. Approximation algorithms are then introduced.

The topics selected are relevant and important for the digital transformation age which is still lacking theoretical treatments. The author has made an attempt to fill this gap and stimulate interest.

Graduate students will not only learn from the materials covered, but also from the research approaches taken in this book.

Professor Kanchana Kanchanasut
School of Engineering and Technology
Asian Institute of Technology

Foreword

It is my great pleasure to write this foreword. In his latest book, *Problem Solving in Algorithms: A Research Approach*, Dr. Sanpawat Kantabutra provides an innovative and refreshing perspective on how to go about solving a problem in a computing discipline. To the extent possible, he walks the reader through the equivalent of the scientific method for computing. Although the work is largely abstract, Dr. Kantabutra does a good job in making it feel concrete and hands-on. His approach is a nuts-and-bolts one, where sometimes a hypothesis leads to a dead end. And, as Dr. Kantabutra points out, such dead ends are part of the process of doing successful research. Along the way to solving difficult problems, failures are encouraged and to be expected. The road to achieving the final answer is a long and winding one, but a very rewarding journey in the long run. This blue-collar approach to critical thinking and problem solving will be of great benefit to many students, researchers, and faculty members alike.

I applaud Dr. Kantabutra for including chapters on an extremely diverse set of topics. He includes material on networking, graph labeling, wireless communication, hierarchical clustering, cyber-security, partitions, graph theory, parallel computation, and clustering. In all cases, he makes a serious effort to provide the reader with the background necessary to comprehend these different domains. Along another dimension, he explores algorithms from a wide variety of angles—sequential, parallel (on several different models), and randomized. Each different topic, combined with a different algorithmic framework, provides him with a rich playing field; one where he can touch on many aspects problem-solving issues and critical-thinking skills. Dr. Kantabutra takes full advantage of all the various options to cover a wide range of techniques and thought processes. Each different model that he describes gives the reader the opportunity to experiment in a broad range of settings.

The book fulfills Dr. Kantabutra's goals of teaching the reader how to problem solve and go about conducting research in any computing domain. The techniques introduce the reader to new ways of thinking and methods for viewing problems. He helps the reader develop a problem-solving toolkit, as it were. When a reader gets stuck in an effort to solve a problem, this book contains techniques that can help guide research or help one to shift directions, say by trying a new process or methodology or asking another relevant question. Dr. Kantabutra takes the reader on a problem-solving journey, and one feels as though he is there accompanying you on that journey—providing encourage-

ment, guidance, strategies, wisdom, and support. The book can help boost one's confidence in going about research. I strongly recommend the book to advanced undergraduate students with a foundation in a computing discipline, graduate students, instructors, and professors who want to get involved in research. The book is well-written, friendly, easy-to-use, and highly innovative. Kudos to Dr. Kantabutra for taking on another ambitious project and seeing it through to an excellent conclusion.

Dr. Raymond Greenlaw
July 20, 2020
Office of Naval Research Distinguished Chair in Cyber-Security, Retired
United States Naval Academy

Preface

This *Problem Solving in Algorithms* book is designed to illustrate how research in theoretical computer science is typically conducted. In particular, some commonly found proof techniques in the design and analysis of algorithms and some important mathematical ideas are discussed in length. Unsurprisingly, our approach in composing this book is research-based because we feel that it is the best way to practice critical thinking and problem-solving skills. Topics in each chapter are chosen such that they are diverse, interesting, and recent. Each chapter can also be used separately as a case study in class. There are ten chapters in this book, covering topics ranging from data clustering, to cyber security, to wireless communication, to graph theory, and to parallel computation. All materials in these chapters are selected from a pool of our recent research publications. In each chapter we first discuss how a theoretical model is defined and the reasons that it is defined in a particular way. We then show how a problem is defined with respect to the theoretical model. Possible proof techniques and the reasons that they might be or might not be used are surveyed. Additionally, an alternative is shown when one is available. In illustrating our points we discuss possible solutions. Of course, some of them might be wrong. But this is what usually happens in research or in problem solving. In fact, it is our intention to show several wrong solutions to a problem because to realize that something is wrong is an essential part of research and self discovery. At the end of each chapter we provide a problem set. A problem marked with an asterisk may be either open, difficult, or time-consuming. This book aims at graduate students, researchers, and computer professionals in computer science or related fields. We expect that readers should have a strong background in the design and analysis of algorithms and discrete mathematics at the undergraduate level. Readers should feel comfortable with basic mathematical proof techniques such as mathematical induction, proof by counter examples, and proof by contradiction. This book is not an introduction to complexity and algorithms. However, when there are certain things that we feel readers might not already have learnt in class, a review and/or a book reference are given.

Dr. Sanpawat Kantabutra
The Theory of Computation Group
July 20, 2020

Contents

List of Figures

List of Tables

Chapter 1

Network Embedding

"When you get stuck, do it over."

In this chapter we introduce a concept of *network embedding*. Roughly speaking, network embedding is a way to use another network A when your physical network is not A. For instance, if your physical network is a ring network but your communication protocol is based on a line network, how are you going to make your communication protocol work without buying a new expensive line network or totally changing the communication protocol to suit a ring network? The first thing to observe is that a line network is a subnetwork of a ring network. Hence, if we could "designate" the edges in the ring network to match those of the line network somehow, our line-based communication protocol would work. However, if our physical network is a hypercube network and our communication protocol is based on a tree network, it is not clear whether a tree network is a subnetwork of a hypercube network. Indeed, if a tree network has many nodes, it is not a subnetwork of a hypercube network. In this case we could still "designate" the edges but with some slowdown factor in speed. For the rest of this chapter we consider a new theoretical network called *Completely Overlapping Network* (CON) and we want to embed a d dimensional hypercube network into a completely overlapping network. Additionally, taking advantage of faster and more powerful processors and a large gap between communication and computation speeds, we also show that the embedded d-dimensional hypercube can be scaled using latency hiding. Related mathematical properties are also shown for optimal scaling. This chapter elucidates the ideas and thinking processes in the original work by Kantabutra and Chawachat [23].

1.1 Introduction

Embedding problems are well known among graph theorists, mathematicians, as well as computer scientists. Typically, embedding problems are described in

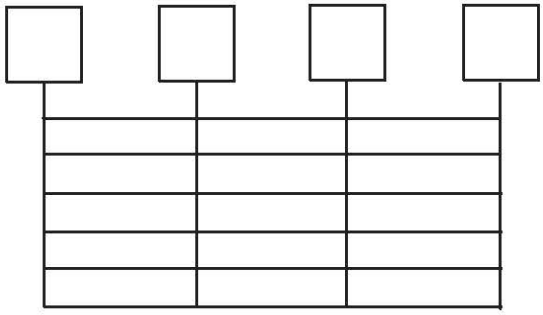
graph theory terms but in our context we describe in computer network terms. There is a lot of motivation behind embedding parallel architectures [1], one of which is the ability to allow one network to operate efficiently in another network with different architectures. Suppose we have two networks A and B. If our physical network is A and we want to run our communication protocol based on B, we "embed" network B into network A. Network A is called an *embedding* or *host* network and network B is called an *embedded* or *guest* network. In other words, we simulate network B in network A.

We consider a binary hypercube in this chapter. The binary hypercube is one of the most versatile and efficient interconnection networks yet discovered for parallel computation. One of the biggest reasons for the popularity of the hypercube is its ability to being both host and guest networks efficiently. In terms of being a host network, a lot of research has been carried out on embedding different topologies into the hypercube. Following are some examples. In [2] and [3] a hypercube is embedded with hierarchical networks. In [4]-[6] authors show how to embed a few kinds of grids into hypercubes. Many kinds of trees can also be embedded into hypercubes as in [7]-[10]. Others embed star networks into hypercubes [11], or shuffle networks into hypercubes [12]. In [13] authors show how to embed a hyper-pyramid into a hypercube. On the other hand, the hypercube can also be a *guest* network. Many parallel algorithms use hypercubes as the communication topology among their processes. Some authors, for instance, try to embed hypercubes into toruses and rings as in [14]-[15]. Others try to embed a hypercube into a mesh as in [16]. In a more mathematical flavor Cayley graphs are shown to be a host for hypercubes in [17]. In area of optical networks, an efficient embedding of a hypercube in a Wavelength Division Multiplexing network is shown in [18].

Motivated by faster Ethernet lines and more powerful processors of the past two decades, a group of computer scientists developed an experimental network called *overlapping network* [19]-[21]. Since connecting several computer nodes to a single Ethernet line limited the shared communication channel, they worked on the concept of using multiple Ethernet lines in some certain configurations. These configurations are in the general classification of overlapping connectivity networks. Overlapping connectivity networks have the characteristic that regions of connectivity are provided and the regions overlap so as to provide parallelism. To generalize their overlapping network for the purpose of our study, more line segments have been added to their overlapping network for a complete parallel connectivity; hence, the name completely overlapping network. A four-node completely overlapping network is shown in Figure 1.1.

At this point several questions should arise. What does an n-node completely overlapping network look like? How many line segments are there in terms of n? How do two nodes communicate in an n-node completely overlapping network? What is the communication protocol? How do we compute time and communication complexities? Because this *is* our research, we have to think about the answers to these questions and eventually give them a model and related definitions.

Figure 1.1: Four-node completely overlapping network



1.2 CON and its Related Definitions

Generally, we want our theoretical model and definitions to reflect reality but there is always a tradeoff to consider. If we design our model to truly reflect reality, we probably have to add every single detail into our model. Every bit of details increases the complexity of our model. The complexity might be so much that we get bogged down in details and are not able to see the whole picture. On the other hand, if our model is too simple, we might not be able to gain insights into anything. Hence, a balance has to be made when we create a model and its related definitions.

We begin with defining an n-node completely overlapping network. Traditionally, a network is described in a graph theory term. A graph has two components. A graph $G = (V, E)$ is a set V of vertices and a set E of edges. If we are going to use a graph G to represent CON, how do we relate V and E to CON? After some thoughts, we realize that two different kinds of vertices are needed because an n-node CON contains non-computer and computer vertices. If we are going to proceed in this direction, we would have $V = V_{\diamond} \cup V_{\triangle}$ and we would have to differentiate the two types of vertices in our communication protocol every time. It seems more complex and tedious than necessary. Can we use some other simpler representation that still suits our purpose? Yes, we can. Because a completely overlapping network is composed of several overlapped communication line segments that connect among several nodes or processors to provide parallelism as shown in Figure 1.1, we can define the network in terms of vertical and horizontal communication line segments. What else do we need to consider putting it in our definition of CON besides the line segments and nodes? None. Hence, we have everything at this point and we define a completely overlapping network using a 3-tuple containing nodes, horizontal line segments, and vertical line segments. The following is the formal definition of an n-node completely overlapping network.

Definition 1 (Completely Overlapping Network). *A* completely overlapping network *is a 3-tuple* (N, H, V)*, where* N*,* H*, and* V *are all finite sets, and*

1. N *is the set of* n *nodes,*

2. *H is the set of $\frac{n(n-1)^2}{2}$ horizontal line segments,*

3. *V is the set of $\frac{n^2(n-1)}{2}$ vertical line segments,*

where H and V constitute a grid-like network.

Several remarks are in order. First, putting the three components into a tuple is a good practice. It forces us to consider only things that are related. Second, N, H, and V cannot be infinite sets in this context. Hence, it is also a good practice to state it as such clearly. Since each of the sets is finite, it is also a good practice to quantify each of them. Observe that horizontal and vertical line segments should be defined in terms of the variable n to show their relationships to the number of nodes because each type of line segments grows with the number of nodes. Lastly, we should make sure that the number of horizontal line segments is indeed $\frac{n(n-1)^2}{2}$ and the number of vertical line segments is indeed $\frac{n^2(n-1)}{2}$ by mathematical induction (or at least do it in our head).

We next have to think about how an n-node CON operates. This stage is relatively easy if we have taken a network course. Our rules of operations here mirror those of the real networking protocols. These rules are reasonable and can certainly be implemented.

Definition 2 (Rules of Operations). *The rules are as follows.*

1. *Horizontal and vertical line segments cannot be shared. In other words, any line segment can be used by only one communication at a time.*

2. *Each line segment is bidirectional.*

3. *Each node has a constant memory size.*

4. *A same message can be concurrently sent from one node to several destination nodes as long as there is no collision of messages.*

5. *If there exists contention for a communication line segment, some kind of priority can be applied.*

Once we have defined the rules of operations and the definition of an n-node completely overlapping network. We need to think about the concept of time. In general, the kinds of time we are interested in involve communication and computation. However, in our context of embedding, we are only interested in communication time. How do we measure it? Obviously, we cannot measure it in real time but at least our choice should reflect real time. The number of horizontal and vertical line segments from node A to node B are proportional to the real time for the communication between A and B. We could certainly use it. But is it too cumbersome? Can we simplify the definition of our communication time while still having a good representation to the real world? In real world vertical line segments are usually within a computer node while horizontal line

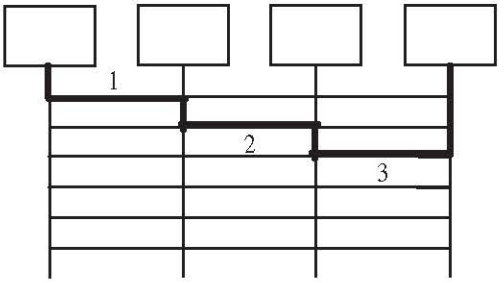


Figure 1.2: Communication time steps

segments are between computer nodes. Hence, the communication time between computer nodes are far more significant than the time within a computer node. Taking this fact into account, we have the following definition.

Definition 3 (Communication Time). *Communication time from one node A to another node B is defined by the number of horizontal line segments the message has to go through from node A to node B.*

Figure 1.2 illustrates a communication between the leftmost node and the rightmost node. This communication takes three communication steps (or three horizontal line segments). At this point our model seems to sufficiently reflect the real world but we should be prepared to revise it as we proceed to do our research. This is often the case where we encounter something that needs to be refined or revised in later stage of research.

Consider next the concept of network embedding. Figure 1.3 shows a two-dimensional hypercube network or a guest network. We want to "embed" it into a four-node CON in Figure 1.1. In other words, we want to *simulate* a two-dimensional hypercube network in a four-node CON and ideally lose nothing in efficiency. What are the components involving this process? Obviously, the nodes of the two networks have to be considered and so do their edges. How do we identify the nodes and the edges? We need to map *both* nodes *and* edges. What is a mathematical concept that we can use to define this whole process? What else do we have to define? Of course, efficiency. We need to be clear about efficiency and quantify it.

To illustrate the embedding concept, consider Figure 1.4. Figure 1.4 illustrates *one* way of embedding a two-dimensional hypercube into a completely overlapping network. We number the nodes from left to right and the lines from top to bottom. The symbol $\otimes$ indicates the beginning and the end of a connection. A connection is represented by a boldfaced line. A path is denoted by $nodeID \Leftrightarrow nodeID$. That is, $00 \Leftrightarrow 01$ is the path between nodes 00 and 01. A notation <line ID; node ID, node ID> is used when a line segment indicating a connection between two nodes is referred to. Let $\doteq$ denote equivalence. In figure 1.4 there are four paths; each path comprises at least one line segment. The four paths and their corresponding segment components

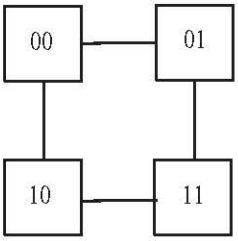


Figure 1.3: Two-dimensional hypercube

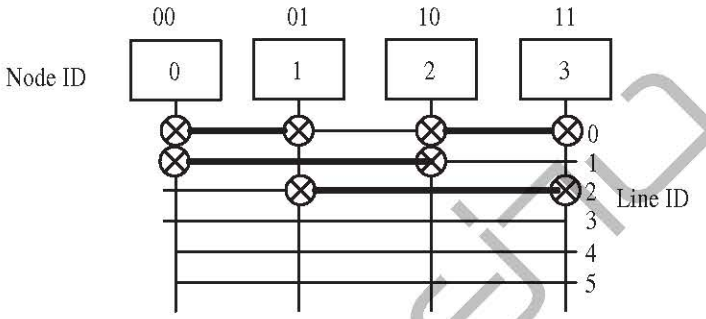


Figure 1.4: Two-dimensional hypercube embedded in the overlapping network

are $00 \Leftrightarrow 01 \doteq \{< 0;00,01 >\}$, $10 \Leftrightarrow 11 \doteq \{< 0;10,11 >\}$, $00 \Leftrightarrow 10 \doteq \{< 1;00,01 >, < 1;01,10 >\}$, and $01 \Leftrightarrow 11 \doteq \{< 2;01,10 >, < 2;10,11 >\}$. Observe that whenever there is an edge in the hypercube there is a corresponding path in CON. Hence, the simulation of the hypercube can be done in CON. Because the communication between any pair of adjacent nodes in hypercube is done through a single edge, we have a slow down factor of two in the embedded hypercube since we have a corresponding path of at most two edges. This fact can be used to define efficiency. Also observe that there are other ways of embedding these two four-node networks as well. What are they?

The term *embedding* is traditionally used with static networks or networks in which all nodes have direct fixed physical links between them. Mesh, torus, and hypercube networks are static but our completely overlapping network is not. In static networks embedding describes mapping nodes of one network onto another network. However, in our problem, we must also choose paths of communication between nodes since we map a static hypercube network onto a dynamic completely overlapping network. The term *dilation* is used to indicate the quality or efficiency of the embedding [1]. The dilation is the length of the longest path in the "embedding" network, (i.e., completely overlapping network) corresponding to one link or edge in the "embedded" network (i.e., hypercube network). When the dilation is one, algorithms would have the same speed in the embedding network as they do in the original network. But it is sometimes

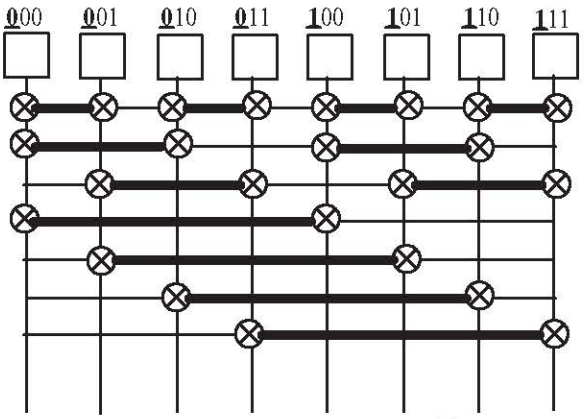


Figure 1.5: Three-dimensional hypercube embedded in the overlapping network

not possible to achieve. In that case, we would like to have a dilation as small as possible so that we will have a small slowdown factor. In Figure 1.4 the dilation is therefore two. At this stage we are ready to formalize the embedding concept.

Definition 4 (Embedding Problem). *Let $C = (P, E)$ be a d-dimensional hypercube, where P is a set of vertices, and E is a set of edges. Let $O = (N, H, V)$ be a completely overlapping network, where N is a set of nodes of an equal size to the set P, H is a set $\{h_1, h_2, h_3, \ldots, h_{\frac{n(n-1)2}{2}}\}$ of horizontal line segments, and V is a set $\{v_1, v_2, v_3, \ldots, v_{\frac{n^2(n-1)}{2}}\}$ of vertical line segments. The optimal embedding problem of C into O is defined by a bijective function $f : P \to N$ and an injective function $g : E \to \mathcal{P}(H)$, where $\mathcal{P}(H)$ is a power set of H. The problem is to minimize the size of the maximal set $x \in \mathcal{P}(H)$ or path x that corresponds to an edge in E.*

Several remarks are in order. Like the definition of an n-node CON, we use a tuple to define a hypercube. Also observe that the naming of things should be easy to remember such as E for edges and H for horizontal line segments unless we have no other obvious choices. Because the embedding is essentially a mapping of vertices and paths, we use the concept of mathematical functions to define it. The mapping between vertices of the two networks is one-to-one and onto while the mapping between the edges and the paths is one-to-one. Hence, we appropriately use bijection and injection, respectively. Among all of these possible mappings, we want a mapping that minimizes a path x that corresponds to an edge in E. When we work with minimization and maximization problems, it is always a good practice to think about the uniqueness of their solutions. For example, we could have two or more ways of embedding that share the same minimum solution.

1.3 The Embedding Algorithm

Given an n-node hypercube network and an n-node CON, we consider ways of embedding the hypercube into the CON. We could use a greedy approach to locally minimize the dilation. For example, we could pick an edge $\{p, p'\} \in E$ of the hypercube network and map the corresponding pair p and p' to nodes i and j in CON such that the dilation between the two nodes i, j in CON is minimal. We repeat the process greedily for the subsequent edges in E until no edge is left. Unfortunately, this greedy approach does not always work (why?). We want to minimize the dilation of the embedding. Can we use a naive approach? Observe that this problem is equivalent to the problem of embedding an n-node hypercube into an n-node line. Hence, a naive approach is to try $n!$ possible ways of mapping the n nodes and pick the one that yields the minimum dilation. Sadly, although a minimum dilation is guaranteed, this approach is not so good because $n! > 2^n$. The time complexity would be too great to wait for the result. Fortunately for us, Harper [22] developed an optimal embedding scheme that embeds a 2^d-node hypercube (or d-cube) into a 2^d-node line. His algorithm ordered the labels of the nodes by their weights. The weight of a label is just the number of one's in its binary representation. Labels with the same weight are ordered in descending order. Then the processes of the 2^d-node hypercube ordered in that way are allocated to the nodes of the line from left to right. However, his embedding solution does not accommodate scaling (or folding) on CON well since it is not symmetric. We will study scaling shortly in the next section.

In the following paragraphs, we describe an algorithm that *partially* solves the embedding problem of a hypercube into a completely overlapping network. Note that this algorithm does not always produce a minimum dilation. The definition of the algorithm is firstly given as follows.

Algorithm **EMBEDDING**(d,CON)
1. If (d=1)
2. connect the two nodes
3. else
4. **CONNECT**(**EMBEDDING**($d-1$,CON),**EMBEDDING**($d-1$,CON))

The inputs to the algorithm are a 2^d-node CON that has not yet been communication-designated and a hypercube dimension d. Subroutine **CONNECT**(...) connects two nodes whose $d-1$ lower-order bits are precisely the same and whose node positions are in different embedded hypercubes. Therefore, each pair of nodes will eventually differ exactly one bit in the d-dimensional embedded hypercube. Figure 1.5 shows the output of the algorithm for embedding a three-dimensional hypercube. Figure 1.6 gives an example of a recursive definition of our hypercube embedding algorithm.

A few remarks are in order. When we describe an algorithm, we always have two options. We could do it iteratively or recursively. Sometimes either

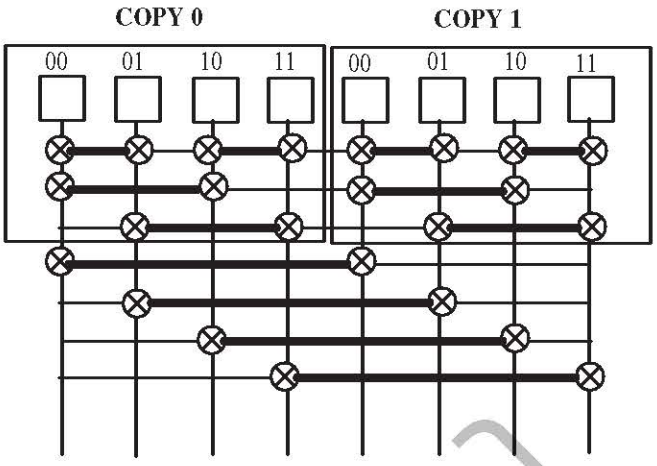


Figure 1.6: Example of the recursive definition

way is fine. But in our case we chose a recursive approach for several reasons. First, a n-node hypercube is recursive by definition. Second, recursive code is usually a lot shorter as it is in this case. Third, recursive code is easier to prove correctness since it is applicable to mathematical proof by induction. It is probably a good brain exercise to write the iterative version of this recursive code, however.

In Figure 1.4 and Figure 1.5 we see that the dilations are 2 and 4 respectively for hypercubes of dimensions 2 and 3. Can we generalize this result? What variables are involved in this generalization? We know that $n = 2^d$. In this case the variable should be d because it is the dimension of the hypercube in consideration and d increases continuously while the value of n does not. For instance, the value of n cannot be 3. What else do we know? In practice we want to gather as many facts that are related to our problem at hand as possible before we attempt to prove it. Some of these facts might be previously proved claims or definitions or some other common mathematical facts or rules. How do we know that we have gathered every fact? Unfortunately, we do not know until the proof is finished. Occasionally, some of these facts may come from a sub-statement that we have just proved. In other words, while we prove things, new facts are generated and some of these new facts will be used later in the context of a larger proof. In our case at hand, we have the definitions of the d-dimensional hypercube and the embedding algorithm and some binary arithmetic. It is also very important to state clearly and unambiguously the claim that we want to prove. After some thoughts and several scratch papers, we have the following proposition.

Proposition 1 (Dilation Size). *The embedding algorithm produces a dilation of 2^{d-1} where d is the hypercube dimension.*

Proof. By the hypercube definition, a d-dimensional hypercube is composed of two $(d-1)$-dimensional hypercubes. Let us call a $(d-1)$-dimensional hypercube with a leading 0 in each binary ID a copy 0 (i.e.,0xxxxx...) and, likewise, a $(d-1)$-dimensional hypercube with a leading 1 in each binary ID a copy 1 (i.e., 1xxxxx...). By definition of the hypercube, each node in copy 0 must have a connection path to its corresponding node in copy 1 (see figure 1.6). Because the nodes in the completely overlapping network are in an increasing order, the dilation of any pair of corresponding nodes is the difference between the two node ID numbers. Since all $d-1$ lower-order bits of the two node ID numbers are exactly the same, the difference is 1 at the bit position d which has a value of 2^{d-1}. □

The algorithm by Harper [22] produces a dilation, for any $d \geq 4$, which is *strictly* less than that of ours (though it is still exponential in d) but we will use our embedding algorithm which is more suitable to scaling as we will see shortly.

1.4 Scaling The Embedded Hypercube

Proposition 1 delivers us a piece of bad news. If the dilation is exponential in d, we can only simulate a small hypercube in CON because of the exponential slowdown. How can we mitigate this problem? With the advent of more powerful coarse-grained processors, computation speed is much higher than communication speed. We want to take advantage of this fact. This method is called *latency hiding* [24]. Since the processors are very fast, if we could allocate two or more processors in the hypercube network to a processor in CON, the communication among all these *virtual* processors within a *physical* processor would become zero. But the question is at what cost? This question is hard to answer at this moment. If we simulate a number of virtual processors by a single physical processor, we do have a decrease in communication time but we also have an increase in computation time. We seem to have a trade-off and we want to precisely *quantify* this trade-off and this is the place where mathematical reasoning becomes very useful as we will illustrate it in the next section. For now, let us consider a latency hiding example of embedding a four-dimensional hypercube into a 16-node CON. We will refer to the combining of at least two nodes into one node as *folding* or *scaling*.

Figure 1.7 shows the original embedding. Figure 1.8 shows the folding of 2 nodes into one. In other words, we use one processor to simulate two processors. Figure 1.9 and Figure 1.10 illustrate further reduction of the dilation and the increase in the number of virtual processors to be simulated by a physical processor. Observe that Figure 1.10 achieves the maximum reduction of the dilation for the 16-node hypercube. When the hypercube has more nodes, things will get more complicated as we continue to fold. We want to precisely describe and represent this process. As usual, several questions should arise in our mind. What variables or quantities are involved in this folding process? How are they

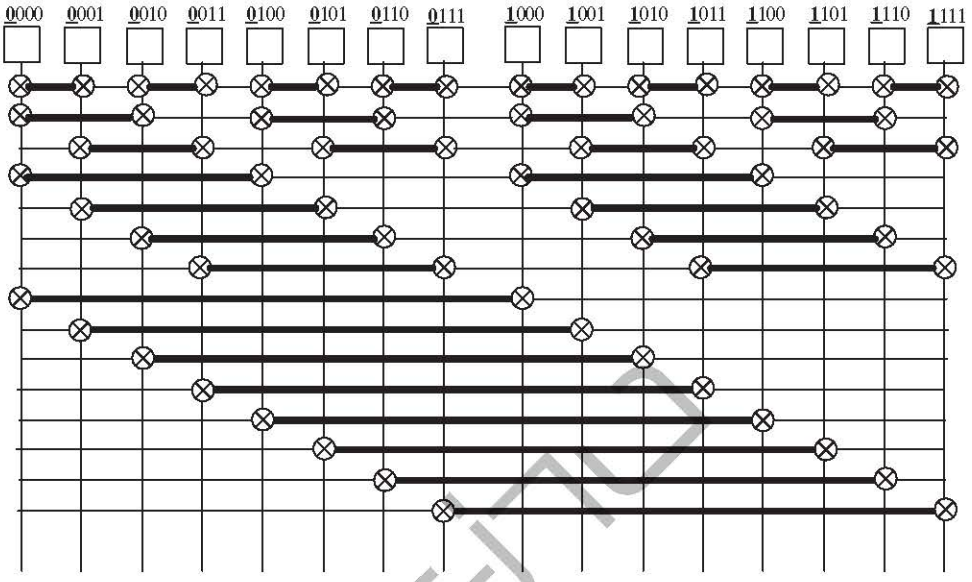


Figure 1.7: An embedded 4-dimensional hypercube in 16-node CON (L(0)=1)

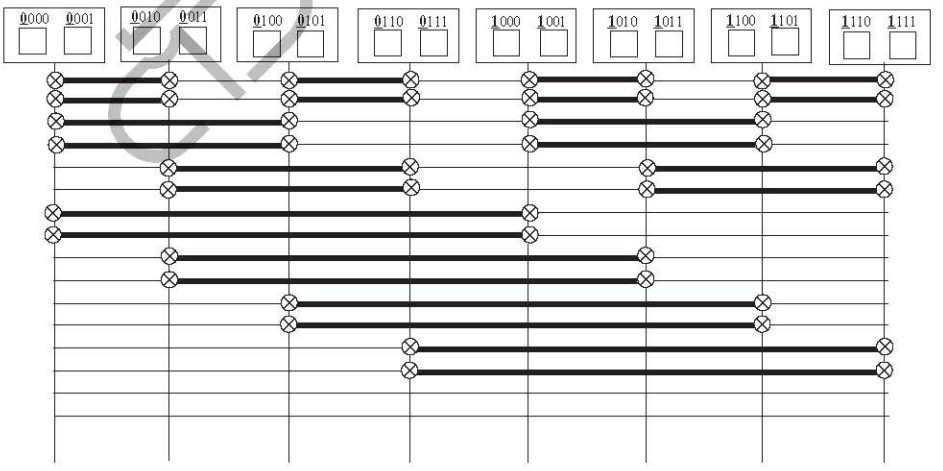


Figure 1.8: A folded CON-embedded 4-dimensional hypercube in 8-node CON with $L(1) = 2$

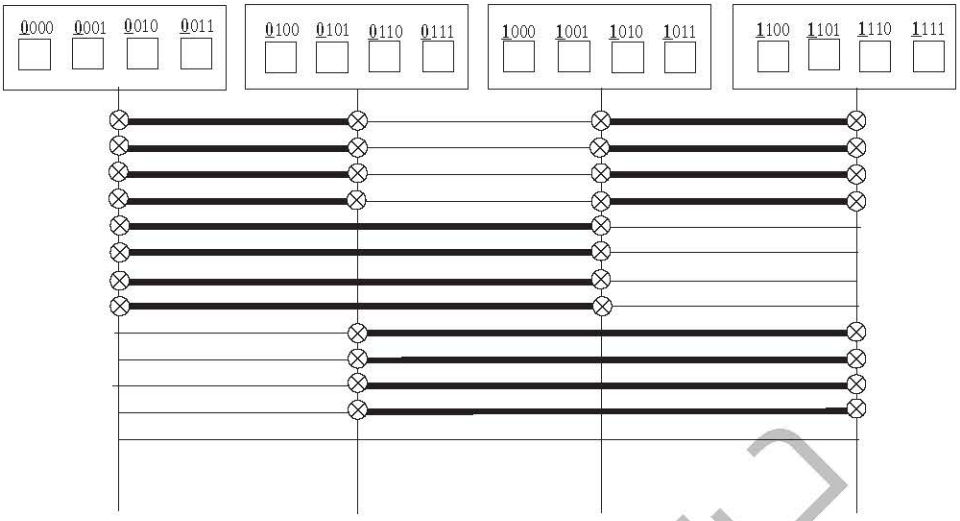


Figure 1.9: A folded CON-embedded 4-dimensional hypercube in 4-node CON with $L(2) = 4$

related? We want to create a mathematical model that reflects reality as much as possible.

By this example, we realize that both dilation and the number of virtual processors to be simulated by a physical processor depend directly on the number of folding times. We also know that dilation and the number of virtual processors are inversely related. In other words, as one grows, the other shrinks. Therefore, we should accordingly name all these quantities as functions of folding times. Let t be the number of folding times and $L(t)$ be the size of the folding or the size of the virtual processors to be simulated by one coarser-grain processor. Let $D(t)$ be the dilation. We also define the value of t to be $0 \leq t \leq d-1$ and initially $L(0) = 1$ and the initial dilation size $D(0)$ is 8 by proposition 1.

At this point we have gained a lot of information and insight from our observation but it is still *not precise*. For instance, we mentioned earlier that the dilation and the number of virtual processors are "inversely related". What exactly does this mean? Can we describe it mathematically? Of course, we can and will see it shortly in the next section.

1.5 The Analysis

In this section we are going to translate our information and intuition gained earlier into mathematics that will enable us to see the relationship of all factors. We knew in the previous section that this mathematics involves folding times t, dilation $D(t)$, and latency $L(t)$. How are we going to describe the relationship of all these factors? Where should we begin to think about it? Remember that

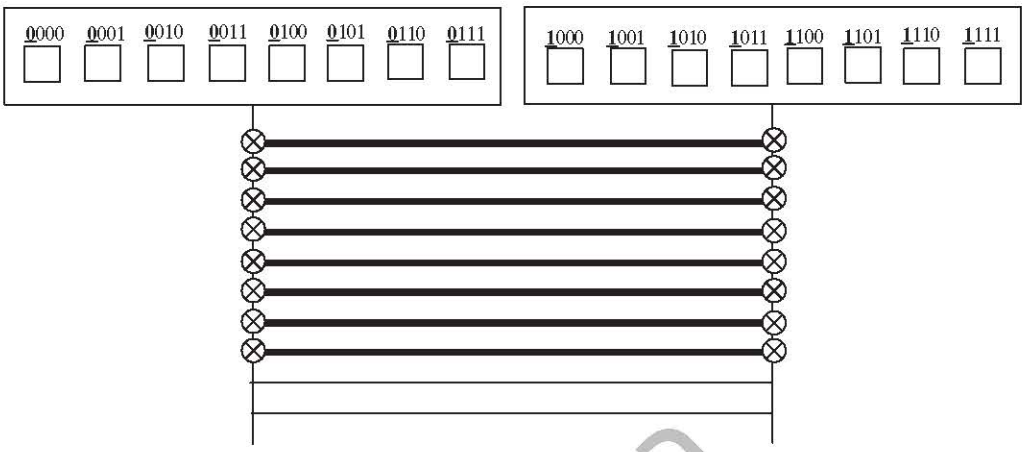


Figure 1.10: A folded CON-embedded 4-dimensional hypercube in 2-node CON with $L(3) = 8$

our goal is to see whether the CON-embedded hypercube is as "good" as the traditional hypercube. What exactly is "good" in this context? If the CON-embedded hypercube is as good as the traditional hypercube, it should have the same performance or time complexity for the same algorithm. Hence, we start from here.

Let $\mathcal{T}^P_{old}$ be the time complexity of the traditional hypercube and $\mathcal{T}^P_{new}(t)$ be the time complexity of the CON-embedded counterpart. Observe that we have factored in the information that the time complexity of the CON-embedded hypercube depends on folding times t into $\mathcal{T}^P_{new}(t)$. We now want to describe each notational symbol mathematically. Before we do, we have an important assumption about the algorithm in the original hypercube. We assume that each node in the hypercube runs the same algorithm. This is a realistic assumption and will make our life easier in the analysis part.

We first describe the time complexity of the original hypercube $\mathcal{T}^P_{old}$. We know that the time complexity in the original hypercube has nothing to do with t, $D(t)$, or $L(t)$. What else do we know? We also know that the time in parallel computation is divided into two parts: communication and computation. Let $\mathcal{T}^{comm}_{old}$ be the communication time complexity and $\mathcal{T}^{comp}_{old}$ be the computation time complexity. Hence, we have the following equation.

$$\mathcal{T}^P_{old} = \mathcal{T}^{comp}_{old} + \mathcal{T}^{comm}_{old} \tag{1.1}$$

Observe that all of the terms in 1.1 are simply notational symbols that tell us about the components of the time complexity of the original hypercube. Can we do it better? Can we be more precise? We additionally know that the computation time of each node is a function $f(n)$ of the input size n. Therefore, we factor in $f(n)$ in 1.1 and have a new equation.

$$\mathcal{T}_{old}^{P} = \mathcal{T}_{old}^{comp} + \mathcal{T}_{old}^{comm} = f(n) + \mathcal{T}_{old}^{comm} \tag{1.2}$$

Consider next the time complexity of the CON-embedded hypercube $\mathcal{T}_{new}^{P}(t)$. Certainly, it must involve t, $D(t)$, and $L(t)$. But how does each of them play its role in the time complexity? Once again, we try to gather all the facts. Given a value of t, we know that each physical processor in the CON-embedded hypercube simulates $L(t)$ virtual processors in the original hypercube. Hence, the computational time complexity in each node of the original hypercube should be increased by a factor of $L(t)$ in CON. Moreover, we also know that each edge in the original hypercube is extended to at most $D(t)$ edges in CON and the parallel time complexity depends on the slowest process. Accordingly, given a value of t, the communication time complexity in the original hypercube should be increased by a factor of $D(t)$ in CON. Appropriately, we have the following equation.

$$\mathcal{T}_{new}^{P}(t) = L(t)f(n) + D(t)\mathcal{T}_{old}^{comm} \tag{1.3}$$

Now that we have equations (1.2) and (1.3). How are we going to learn anything from them? After all, they are just facts. Remember that our goal is to see whether the CON-embedded hypercube is as *good* as the traditional hypercube. Can we, for example, do the following? Consider equation (1.3) - equation (1.2).

$$\mathcal{T}_{new}^{P}(t) - \mathcal{T}_{old}^{P} = L(t)f(n) + D(t)\mathcal{T}_{old}^{comm} - (f(n) + \mathcal{T}_{old}^{comm}) \tag{1.4}$$

If equation (1.4) is evaluated to zero, we know that our CON-embedded hypercube is just as good as the original hypercube. If equation (1.4) is evaluated to a negative quantity, we know that our CON-embedded hypercube is that much better than the original hypercube. If equation (1.4) is evaluated to a positive quantity, we know that our CON-embedded hypercube is that much worse than the original hypercube. Equation (1.4) seems really nice but it does not tell us in terms of a factor. It would be a lot nicer or more descriptive if we can say our CON-embedded hypercube is 10 times faster than the original hypercube, for instance. Hence, instead of subtraction, we use division. In other words, we use the ratio $\frac{\mathcal{T}_{old}^{P}}{\mathcal{T}_{new}^{P}(t)}$ and name it *scaling indicator*.

We have come along very far at this point from nothing to the scaling indicator. But look very carefully at equation (1.1). What are we adding? Are we adding apples and oranges? In other words, does one unit of computation time equate one unit of communication time. Of course not. We have to fix this issue to reflect reality. Fortunately, fixing can be easily done. We convert computation steps into communication steps. If one communication step equates C computation steps, we can derive two new equations as follows.

$$\mathcal{T}_{old}^{P} = \frac{f(n)}{C} + \mathcal{T}_{old}^{comm} \tag{1.5}$$

$$\mathcal{T}_{new}^{P}(t) = L(t)\frac{f(n)}{C} + D(t)\mathcal{T}_{old}^{comm} \tag{1.6}$$

Note that in reality C does exist. Furthermore, we also assume that $\frac{f(n)}{C} \ll \mathcal{T}_{old}^{comm}$, which is reasonable in reality.

Ideally, we would like our folded CON-embedded hypercube to perform at least as well as the original hypercube. The question is whether this is possible. We have not yet proved that there exists a way to embed a d-dimensional hypercube into a CON such that the ratio of the edge in the hypercube to the edge in CON is 1:1. After we did several experiments on papers, our intuition informs us that the ratio of 1:1 is impossible, implying that it is not possible that our folded CON-embedded hypercube performs at least as well as the original hypercube. Here is the important part. How are we going to convince others to believe that such embedding is not possible? Experiments are certainly not sufficient. Of course, we have to use mathematical reasoning to show it.

Before we start to do the formal proof, we again try to collect as many related facts as possible. We know t, $\mathcal{T}_{old}^{P}$, $\mathcal{T}_{new}^{P}(t)$, and the scaling indicator $\frac{\mathcal{T}_{old}^{P}}{\mathcal{T}_{new}^{P}(t)}$. Our goal is show that the CON-embedded hypercube is not as *good* as the traditional hypercube. What exactly does it mean? We need to state our claim clearly. More precisely, we need to show that $\mathcal{T}_{old}^{P} \neq \mathcal{T}_{new}^{P}(t)$ for all t such that $0 \leq t \leq d-1$. First, observe that we need to show that this claim is true for all values of t. We also note that t depends on d.

A little bit of experiments might give us some insights. After picking some small value of d and plugging each possible value of t in equations (1.5) and (1.6), our first intuition seems to tell that it is not possible to make the two equations equal. After second and third trials of small values of d, we are convinced that the two equations are not equal. How do we generalize this intuition for all values of d and t? Look at equations (1.5) and (1.6) carefully. The only difference between the two equations is the factors $L(t)$ and $D(t)$ in (1.6). If the two equations are equal, both $L(t)$ and $D(t)$ have to be one simultaneously. Is it possible? As it turns out, it is not possible. We formalize our intuition into a proof.

Proposition 2 (No Exact Simulation). *$\mathcal{T}_{old}^{P} \neq \mathcal{T}_{new}^{P}(t)$ for all $0 \leq t \leq d-1$.*

Proof. From (1.5) and (1.6) $\mathcal{T}_{old}^{P} = \mathcal{T}_{new}^{P}(t)$ when $L(t) = 1 = D(t)$ but this is not possible because $D(t_{max}) = 1 = L(0)$ and $t_{max} = log_2(D(0)) \neq 0$ where $D(0)$ is the initial dilation. Thus, the proposition holds. □

Proposition 2 tells us that *exact* simulation of a hypercube on CON using a folded CON-embedded hypercube is *not* possible. It means that, when one simulates a hypercube in CON using a folded CON-embedded hypercube, some slowdown in speed relative to that of the original hypercube can be expected, regardless of the number of folding times. So the best we can hope for is the simulation with *minimum* slowdown. More specifically, we have to figure out the value of t that enables the minimum slowdown.

Chiang Mai University Press

Chaing Mai University Press, founded in 2009, is a unit of the Office of Research Administration Center. Its objective is to select, publish and disseminate quality academic books and there by support the university's focus on research and academic excellence.

CHIANG MAI
UNIVERSITY PRESS

Office of Research Administration Center
Office of the University
Chiang Mai University
Tel: 66 53 94 3603-5
Fax: 66 53 94 3600
E-mail: cmupress.th@gmail.com
Website: https://cmupress.cmu.ac.th

This is THE book for every serious researcher in theoretical computer science. The book exposes critical detail in problem solving and researching in the fields of algorithms and complexity that no other book has ever done. It reveals the secrets of doing research and the way of thinking that are so natural to the world's top computer scientists. Such skills and thinking are so "second nature" to every top computer scientist that they are not even mentioned or talked about. This book is thus for everyone who seriously wants to become an excellent researcher but may not have such skills and thinking.

ISBN (e-Book): 978-616-398-549-1

CHIANG MAI UNIVERSITY PRESS